

# **Introduction to FreeS/WAN**

# Table of Contents

<b>Introduction.....</b>	<b>1</b>
<u>IPsec, Security for the Internet Protocol.....</u>	1
<u>Interoperating with other IPsec implementations.....</u>	2
<u>Advantages of IPsec.....</u>	2
<u>Applications of IPsec.....</u>	2
<u>The need to authenticate gateways.....</u>	4
<u>The FreeS/WAN project.....</u>	5
<u>Project goals.....</u>	5
<u>Project team.....</u>	6
<u>Products containing FreeS/WAN.....</u>	6
<u>Full Linux distributions.....</u>	6
<u>Linux kernel distributions.....</u>	7
<u>Office server distributions.....</u>	7
<u>Firewall distributions.....</u>	7
<u>Firewall and VPN products.....</u>	7
<u>Information sources.....</u>	8
<u>This HowTo, in multiple formats.....</u>	8
<u>RTFM (please Read The Fine Manuals).....</u>	8
<u>Other documents in the distribution.....</u>	9
<u>Background material.....</u>	9
<u>Archives of the project mailing list.....</u>	10
<u>User-written HowTo information.....</u>	10
<u>Papers on FreeS/WAN.....</u>	10
<u>License and copyright information.....</u>	11
<u>Distribution sites.....</u>	11
<u>Primary site.....</u>	11
<u>Mirrors.....</u>	11
<u>The "munitions" archive of Linux crypto software.....</u>	12
<u>Links to other sections.....</u>	12
 <b>Upgrading to FreeS/WAN 2.x.....</b>	 <b>13</b>
<u>New! Built in Opportunistic connections.....</u>	13
<u>Upgrading Opportunistic Encryption to 2.01 (or later).....</u>	13
<u>New! Policy Groups.....</u>	13
<u>New! Packetdefault Connection.....</u>	14
<u>FreeS/WAN now disables Reverse Path Filtering.....</u>	14
<u>Revised ipsec.conf.....</u>	14
<u>No promise of compatibility.....</u>	14
<u>Most ipsec.conf files will work fine.....</u>	14
<u>Backward compatibility patch.....</u>	15
<u>Details.....</u>	15
<u>Upgrading from 1.x RPMs to 2.x RPMs.....</u>	15
 <b>Quickstart Guide to Opportunistic Encryption.....</b>	 <b>17</b>
<u>Purpose.....</u>	17
<u>OE "flag day".....</u>	17
<u>Requirements.....</u>	17
<u>RPM install.....</u>	17

# Table of Contents

## **Quickstart Guide to Opportunistic Encryption**

<u>Download RPMs</u> .....	17
<u>Check signatures</u> .....	18
<u>Install the RPMs</u> .....	18
<u>Test</u> .....	18
<u>Our Opportunistic Setups</u> .....	19
<u>Full or partial opportunism?</u> .....	19
<u>Initiate-only setup</u> .....	19
<u>Restrictions</u> .....	19
<u>Create and publish a forward DNS record</u> .....	19
<u>Test that your key has been published</u> .....	20
<u>Configure, if necessary</u> .....	20
<u>Test</u> .....	21
<u>Full Opportunism</u> .....	21
<u>Put a TXT record in a Forward Domain</u> .....	21
<u>Put a TXT record in Reverse DNS</u> .....	21
<u>Test your DNS record</u> .....	21
<u>No Configuration Needed</u> .....	22
<u>Consider Firewalling</u> .....	22
<u>Test</u> .....	22
<u>Test</u> .....	22
<u>Testing opportunistic connections</u> .....	22
<u>Now what?</u> .....	23
<u>Notes</u> .....	23
<u>Troubleshooting OE</u> .....	23
<u>Known Issues</u> .....	23

## **How to Configure Linux FreeS/WAN with Policy Groups**.....24

<u>What are Policy Groups?</u> .....	24
<u>Built-In Security Options</u> .....	24
<u>Using Policy Groups</u> .....	25
<u>Example 1: Using a Base Policy Group</u> .....	25
<u>Example 2: Defining IPsec Security Policy with Groups</u> .....	25
<u>Example 3: Creating a Simple IPsec VPN with the private Group</u> .....	26
<u>Example 4: New Policy Groups to Protect a Subnet</u> .....	27
<u>Example 5: Adding a Subnet to the VPN</u> .....	28
<u>Appendix</u> .....	29
<u>Our Hidden Connections</u> .....	29
<u>Custom Policy Groups</u> .....	29
<u>Disabling Opportunistic Encryption</u> .....	29

## **FreeS/WAN FAQ**.....30

<u>Index of FAQ questions</u> .....	30
<u>What is FreeS/WAN?</u> .....	32
<u>How do I report a problem or seek help?</u> .....	32
<u>Can I get ...</u> .....	33
<u>Can I get an off-the-shelf system that includes FreeS/WAN?</u> .....	33
<u>Can I hire consultants or staff who know FreeS/WAN?</u> .....	33

# Table of Contents

## **FreeS/WAN FAQ**

<u>Can I get commercial support?</u> .....	33
<u>Release questions</u> .....	33
<u>What is the current release?</u> .....	33
<u>When is the next release?</u> .....	33
<u>Are there known bugs in the current release?</u> .....	34
<u>Modifications and contributions</u> .....	34
<u>Can I modify FreeS/WAN to ...?</u> .....	34
<u>Can I contribute to the project?</u> .....	34
<u>Is there detailed design documentation?</u> .....	34
<u>Will FreeS/WAN work in my environment?</u> .....	35
<u>Can FreeS/WAN talk to ...?</u> .....	35
<u>Can different FreeS/WAN versions talk to each other?</u> .....	35
<u>Is there a limit on throughput?</u> .....	35
<u>Is there a limit on number of tunnels?</u> .....	35
<u>Is a ... fast enough to handle FreeS/WAN with my loads?</u> .....	35
<u>Will FreeS/WAN work on ... ?</u> .....	36
<u>Will FreeS/WAN run on my version of Linux?</u> .....	36
<u>Will FreeS/WAN run on non-Intel CPUs?</u> .....	36
<u>Will FreeS/WAN run on multiprocessors?</u> .....	36
<u>Will FreeS/WAN work on an older kernel?</u> .....	36
<u>Will FreeS/WAN run on the latest kernel version?</u> .....	36
<u>Will FreeS/WAN work on unusual network hardware?</u> .....	37
<u>Will FreeS/WAN work on a VLAN (802.1q) network?</u> .....	37
<u>Does FreeS/WAN support ...</u> .....	38
<u>Does FreeS/WAN support site-to-site VPN (Virtual Private Network) applications?</u> .....	38
<u>Does FreeS/WAN support remote users connecting to a LAN?</u> .....	38
<u>Does FreeS/WAN support remote users using shared secret authentication?</u> .....	38
<u>Does FreeS/WAN support wireless networks?</u> .....	38
<u>Does FreeS/WAN support X.509 or other PKI certificates?</u> .....	38
<u>Does FreeS/WAN support user authentication (Radius, SecureID, Smart Card...)?</u> .....	39
<u>Does FreeS/WAN support NAT traversal?</u> .....	39
<u>Does FreeS/WAN support assigning a "virtual identity" to a remote system?</u> .....	39
<u>Does FreeS/WAN support single DES encryption?</u> .....	40
<u>Does FreeS/WAN support AES encryption?</u> .....	40
<u>Does FreeS/WAN support other encryption algorithms?</u> .....	40
<u>Can I ...</u> .....	40
<u>Can I use policy groups along with explicitly configured connections?</u> .....	40
<u>Can I turn off policy groups?</u> .....	41
<u>Can I reload connection info without restarting?</u> .....	41
<u>Can I use several masqueraded subnets?</u> .....	41
<u>Can I use subnets masqueraded to the same addresses?</u> .....	42
<u>Can I assign a road warrior an address on my net (a virtual identity)?</u> .....	43
<u>Can I support many road warriors with one gateway?</u> .....	44
<u>Can I have many road warriors using shared secret authentication?</u> .....	44
<u>Can I use Quality of Service routing with FreeS/WAN?</u> .....	45
<u>Can I recognise dead tunnels and shut them down?</u> .....	45
<u>Can I build IPsec tunnels over a demand-dialed link?</u> .....	47

# Table of Contents

## FreeS/WAN FAQ

<u>Can I build GRE, L2TP or PPTP tunnels over IPsec?</u>	47
<u>... use Network Neighborhood (Samba, NetBIOS) over IPsec?</u>	48
<u>Life's little mysteries</u>	48
<u>I cannot ping ...</u>	48
<u>It takes forever to ...</u>	49
<u>I send packets to the tunnel with route(8) but they vanish</u>	50
<u>When a tunnel goes down, packets vanish</u>	51
<u>The firewall ate my packets!</u>	52
<u>Dropped connections</u>	52
<u>Disappearing %defaultroute</u>	53
<u>TCPdump on the gateway shows strange things</u>	53
<u>Traceroute does not show anything between the gateways</u>	54
<u>Testing in stages</u>	55
<u>Manually keyed connections don't work</u>	55
<u>One manual connection works, but second one fails</u>	55
<u>Manual connections work, but automatic keying doesn't</u>	56
<u>IPsec works, but connections using compression fail</u>	56
<u>Small packets work, but large transfers fail</u>	56
<u>Subnet-to-subnet works, but tests from the gateways don't</u>	56
<u>Compilation problems</u>	56
<u>gmp.h: No such file or directory</u>	56
<u>... virtual memory exhausted</u>	57
<u>Interpreting error messages</u>	57
<u>route-client (or host) exited with status 7</u>	57
<u>SIOCADDRT: Network is unreachable</u>	59
<u>ipsec setup: modprobe: Can't locate module ipsec</u>	59
<u>ipsec setup: Fatal error, kernel appears to lack KLIPS</u>	59
<u>ipsec setup: ... failure to fetch key for ... from DNS</u>	60
<u>ipsec setup: ... interfaces ... and ... share address ...</u>	61
<u>ipsec setup: Cannot adjust kernel flags</u>	61
<u>Message numbers (MI3, QR1, et cetera) in Pluto messages</u>	62
<u>Connection names in Pluto error messages</u>	62
<u>Pluto: ... can't orient connection</u>	62
<u>... we have no ipsecN interface for either end of this connection</u>	62
<u>Pluto: ... no connection is known</u>	63
<u>Pluto: ... no suitable connection</u>	63
<u>Pluto: ... no connection has been authorized</u>	64
<u>Pluto: ... OAKLEY DES CBC is not supported</u>	65
<u>Pluto: ... no acceptable transform</u>	65
<u>rsasigkey dumps core</u>	66
<u>!Pluto failure!: ... exited with ... signal 4</u>	66
<u>ECONNREFUSED error message</u>	66
<u>klips debug: ... no eroute!</u>	66
<u>... trouble writing to /dev/ipsec ... SA already in use</u>	68
<u>... ignoring ... payload</u>	68
<u>unknown parameter name "rightcert"</u>	68
<u>Why don't you restrict the mailing lists to reduce spam?</u>	68

# Table of Contents

<b>FreeS/WAN manual pages.....</b>	<b>72</b>
<u>Files.....</u>	72
<u>Commands.....</u>	72
<u>Library routines.....</u>	73
<b>FreeS/WAN and firewalls.....</b>	<b>75</b>
<u>Filtering rules for IPsec packets.....</u>	75
<u>Firewall configuration at boot.....</u>	75
<u>A simple set of rules.....</u>	76
<u>Other rules.....</u>	76
<u>Published rule sets.....</u>	77
<u>Calling firewall scripts, named in ipsec.conf(5).....</u>	78
<u>Scripts called at IPsec start and stop.....</u>	78
<u>Scripts called at connection up and down.....</u>	78
<u>Scripts for ipchains or iptables.....</u>	79
<u>A complication: IPsec vs. NAT.....</u>	80
<u>NAT on or behind the IPsec gateway works.....</u>	80
<u>NAT between gateways is problematic.....</u>	81
<u>Other references on NAT and IPsec.....</u>	81
<u>Other complications.....</u>	81
<u>IPsec through the gateway.....</u>	81
<u>Preventing non-IPsec traffic.....</u>	82
<u>Filtering packets from unknown gateways.....</u>	82
<u>Other packet filters.....</u>	83
<u>ICMP filtering.....</u>	83
<u>UDP packets for traceroute.....</u>	84
<u>UDP for L2TP.....</u>	84
<u>How it all works: IPsec packet details.....</u>	84
<u>ESP and AH do not have ports.....</u>	85
<u>Header layout.....</u>	85
<u>DHR on the updown script.....</u>	86
<b>Linux FreeS/WAN Troubleshooting Guide.....</b>	<b>88</b>
<u>Overview.....</u>	88
<u>1. During Install.....</u>	88
<u>1.1 RPM install gotchas.....</u>	88
<u>1.2 Problems installing from source.....</u>	88
<u>1.3 Install checks.....</u>	88
<u>1.3 Troubleshooting OE.....</u>	89
<u>2. During Negotiation.....</u>	91
<u>2.1 Determine Connection State.....</u>	91
<u>2.2 Finding error text.....</u>	92
<u>2.3 Interpreting a Negotiation Error.....</u>	93
<u>3. Using a Connection.....</u>	93
<u>3.1 Orienting yourself.....</u>	93
<u>3.2 Those pesky configuration errors.....</u>	94
<u>3.3 Check Routing and Firewalling.....</u>	94
<u>3.4 When in doubt, sniff it out.....</u>	95

# Table of Contents

<b><u>Linux FreeS/WAN Troubleshooting Guide</u></b>	
<u>3.5 Check your logs</u>	96
<u>3.6 More testing for the truly thorough</u>	96
<u>4. Problem Reporting</u>	96
<u>4.1 How to ask for help</u>	96
<u>4.2 Where to ask</u>	97
<u>5. Additional Notes on Troubleshooting</u>	97
<u>5.1 Information available on your system</u>	97
<u>5.2 Testing between security gateways</u>	98
<u>5.3 ifconfig reports for KLIPS debugging</u>	99
<u>5.4 Using GDB on Pluto</u>	100
<b><u>Linux FreeS/WAN Compatibility Guide</u></b>	<b>101</b>
<u>Implemented parts of the IPsec Specification</u>	101
<u>In Linux FreeS/WAN</u>	101
<u>Deliberately omitted</u>	102
<u>Not (yet) in Linux FreeS/WAN</u>	102
<u>Our PF-Key implementation</u>	103
<u>PF-Key portability</u>	103
<u>Kernels other than the latest 2.2.x and 2.4.y</u>	103
<u>2.0.x kernels</u>	104
<u>2.2 and 2.4 kernels</u>	104
<u>Intel Linux distributions other than Redhat</u>	104
<u>Redhat 7.0</u>	104
<u>SuSE Linux</u>	105
<u>Slackware</u>	106
<u>Debian</u>	106
<u>Caldera</u>	107
<u>CPUs other than Intel</u>	107
<u>Corel Netwinder (StrongARM CPU)</u>	107
<u>Yellow Dog Linux on Power PC</u>	107
<u>Mklinux</u>	109
<u>Alpha 64-bit processors</u>	109
<u>Sun SPARC processors</u>	110
<u>MIPS processors</u>	110
<u>Transmeta Crusoe</u>	110
<u>Motorola Coldfire</u>	110
<u>Multiprocessor machines</u>	111
<u>Support for crypto hardware</u>	111
<u>IP version 6 (IPv6)</u>	112
<u>IPv6 background</u>	112
<b><u>Interoperating with FreeS/WAN</u></b>	<b>114</b>
<u>Interop at a Glance</u>	114
<u>Key</u>	115
<u>Basic Interop Rules</u>	115
<u>Longer Stories</u>	116
<u>For More Compatible Implementations</u>	116

# Table of Contents

<b><u>Interoperating with FreeS/WAN</u></b>	
<u>For Other Implementations</u> .....	118
<b><u>Performance of FreeS/WAN</u></b> .....	<b>126</b>
<u>Published material</u> .....	126
<u>Estimating CPU overheads</u> .....	126
<u>Higher performance alternatives</u> .....	127
<u>Other considerations</u> .....	128
<u>Many tunnels from a single gateway</u> .....	128
<u>Low-end systems</u> .....	129
<u>Measuring KLIPS</u> .....	130
<u>Speed with compression</u> .....	131
<u>Methods of measuring</u> .....	132
<b><u>Testing FreeS/WAN</u></b> .....	<b>134</b>
<u>Testing opportunistic connections</u> .....	134
<u>Basic OE Test</u> .....	134
<u>OE Gateway Test</u> .....	135
<u>Additional OE tests</u> .....	135
<u>Testing with User Mode Linux</u> .....	136
<u>Configuration for a testbed network</u> .....	136
<u>Testbed network</u> .....	136
<u>Using packet sniffers in testing</u> .....	137
<u>Verifying encryption</u> .....	138
<u>Mailing list pointers</u> .....	139
<b><u>Kernel configuration for FreeS/WAN</u></b> .....	<b>140</b>
<u>Not everyone needs to worry about kernel configuration</u> .....	140
<u>Assumptions and notation</u> .....	140
<u>Labels used</u> .....	140
<u>Kernel options for FreeS/WAN</u> .....	141
<b><u>Other configuration possibilities</u></b> .....	<b>146</b>
<u>Some rules of thumb about configuration</u> .....	146
<u>Tunnels are cheap</u> .....	146
<u>Subnet sizes</u> .....	147
<u>Other network layouts</u> .....	148
<u>Choosing connection types</u> .....	149
<u>Manual vs. automatic keying</u> .....	150
<u>Authentication methods for auto-keying</u> .....	150
<u>Advantages of public key methods</u> .....	151
<u>Using shared secrets in production</u> .....	152
<u>Putting secrets in ipsec.secrets(5)</u> .....	152
<u>File security</u> .....	153
<u>Shared secrets for road warriors</u> .....	153
<u>Using manual keying in production</u> .....	154
<u>Creating keys with ranbits</u> .....	156
<u>Setting up connections at boot time</u> .....	156

# Table of Contents

## **Other configuration possibilities**

<u>Multiple tunnels between the same two gateways</u> .....	158
<u>One tunnel plus advanced routing</u> .....	159
<u>An Opportunistic Gateway</u> .....	160
<u>Start from full opportunism</u> .....	160
<u>Reverse DNS TXT records for each protected machine</u> .....	160
<u>Publish your records</u> .....	161
<u>...and test them</u> .....	161
<u>No Configuration Needed</u> .....	161
<u>Extruded Subnets</u> .....	162
<u>Road Warrior with virtual IP address</u> .....	163
<u>Dynamic Network Interfaces</u> .....	165
<u>Basics</u> .....	165
<u>Boot Time</u> .....	166
<u>Change Time</u> .....	166
<u>Unencrypted tunnels</u> .....	166

## **Installing FreeS/WAN**.....168

<u>Requirements</u> .....	168
<u>Choose your install method</u> .....	168
<u>FreeS/WAN ships with some Linuxes</u> .....	168
<u>FreeS/WAN may be altered</u> .....	168
<u>You might need to create an authentication keypair</u> .....	168
<u>Start and test FreeS/WAN</u> .....	169
<u>RPM install</u> .....	169
<u>Download RPMs</u> .....	169
<u>For freeswan.org RPMs: check signatures</u> .....	170
<u>Install the RPMs</u> .....	170
<u>Start and Test FreeS/WAN</u> .....	171
<u>Install from Source</u> .....	171
<u>Decide what functionality you need</u> .....	171
<u>Download FreeS/WAN</u> .....	171
<u>For freeswan.org source: check its signature</u> .....	171
<u>Untar, unzip</u> .....	171
<u>Patch if desired</u> .....	172
<u>... and Make</u> .....	172
<u>Start FreeS/WAN and test your install</u> .....	173
<u>Test your install</u> .....	173
<u>Making FreeS/WAN play well with others</u> .....	173
<u>Configure for your needs</u> .....	173

## **How to configure FreeS/WAN**.....174

<u>Requirements</u> .....	174
<u>Net-to-Net connection</u> .....	174
<u>Gather information</u> .....	174
<u>Edit /etc/ipsec.conf</u> .....	175
<u>Start your connection</u> .....	176
<u>Do not MASQ or NAT packets to be tunneled</u> .....	176

# Table of Contents

<b><u>How to configure FreeS/WAN</u></b>	
<u>Test your connection</u>	176
<u>Finishing touches</u>	177
<u>Road Warrior Configuration</u>	177
<u>Gather information</u>	177
<u>Customize /etc/ipsec.conf</u>	178
<u>Start your connection</u>	178
<u>Do not MASQ or NAT packets to be tunneled</u>	179
<u>Test your connection</u>	179
<u>Finishing touches</u>	179
<u>Multiple Road Warriors</u>	180
<u>What next?</u>	180
<b><u>Linux FreeS/WAN background</u></b>	<b>181</b>
<u>Some DNS background</u>	181
<u>Forward and reverse maps</u>	181
<u>Hierarchy and delegation</u>	182
<u>Syntax of DNS records</u>	182
<u>Cacheing, TTL and propagation delay</u>	182
<u>Problems with packet fragmentation</u>	183
<u>Network address translation (NAT)</u>	185
<u>NAT to non-routable addresses</u>	185
<u>NAT to routable addresses</u>	186
<b><u>FreeS/WAN script examples</u></b>	<b>187</b>
<u>Poltorak's Firewall script</u>	187
<b><u>How to configure to use "make check"</u></b>	<b>192</b>
<u>What is "make check"</u>	192
<u>Running "make check"</u>	192
<b><u>How to write a "make check" test</u></b>	<b>193</b>
<u>Structure of a test</u>	193
<u>The TESTLIST</u>	193
<u>Test kinds</u>	193
<u>Common parameters</u>	194
<u>KLIPStest paramaters</u>	195
<u>mkinsttest paramaters</u>	196
<u>rpm build install test paramaters</u>	197
<u>libtest paramaters</u>	197
<u>umlplutotest paramaters</u>	197
<u>umlXhost parameters</u>	197
<u>kernel patch test paramaters</u>	199
<u>module compile paramaters</u>	199
<b><u>Current pitfalls</u></b>	<b>201</b>

# Table of Contents

<b><u>User-Mode-Linux Testing guide</u></b> .....	<b>202</b>
<u>Preliminary Notes on BIND</u> .....	202
<u>Steps to Install UML for FreeS/WAN</u> .....	202
<b><u>Debugging the kernel with GDB</u></b> .....	<b>205</b>
<u>Other notes about debugging</u> .....	205
<b><u>User-Mode-Linux mysteries</u></b> .....	<b>206</b>
<b><u>Getting more info from uml netjig</u></b> .....	<b>207</b>
<b><u>History and politics of cryptography</u></b> .....	<b>208</b>
<u>Introduction</u> .....	208
<u>History</u> .....	208
<u>Politics</u> .....	211
<u>Links</u> .....	212
<u>Outline of this section</u> .....	212
<u>From our project leader</u> .....	213
<u>Swan: Securing the Internet against Wiretapping</u> .....	213
<u>Stopping wholesale monitoring</u> .....	216
<u>Government promotion of weak crypto</u> .....	217
<u>Escrowed encryption</u> .....	217
<u>Limited key lengths</u> .....	218
<u>Cryptography Export Laws</u> .....	220
<u>US Law</u> .....	220
<u>What's wrong with restrictions on cryptography</u> .....	221
<u>The Wassenaar Arrangement</u> .....	222
<u>Export status of Linux FreeS/WAN</u> .....	222
<u>Help spread IPsec around</u> .....	223
<u>DES is Not Secure</u> .....	224
<u>Dedicated hardware breaks DES in a few days</u> .....	224
<u>Spooks may break DES faster yet</u> .....	224
<u>Networks break DES in a few weeks</u> .....	225
<u>We disable DES</u> .....	226
<u>40-bits is laughably weak</u> .....	226
<u>Triple DES is almost certainly secure</u> .....	226
<u>AES in IPsec</u> .....	226
<u>Press coverage of Linux FreeS/WAN:</u> .....	227
<u>FreeS/WAN 1.0 press</u> .....	227
<u>Press release for version 1.0</u> .....	227
<b><u>The IPsec protocols</u></b> .....	<b>229</b>
<u>Protocols and phases</u> .....	229
<u>Applying IPsec</u> .....	229
<u>Advantages of IPsec</u> .....	230
<u>Limitations of IPsec</u> .....	230
<u>IPsec is a general mechanism for securing IP</u> .....	231
<u>Using authentication without encryption</u> .....	232

# Table of Contents

<b><u>The IPsec protocols</u></b>	
<u>Encryption without authentication is dangerous</u>	233
<u>Multiple layers of IPsec processing are possible</u>	234
<u>Resisting traffic analysis</u>	234
<u>Cryptographic components</u>	236
<u>Block ciphers</u>	236
<u>Hash functions</u>	236
<u>Diffie–Hellman key agreement</u>	237
<u>RSA authentication</u>	237
<u>Structure of IPsec</u>	237
<u>IKE (Internet Key Exchange)</u>	237
<u>IPsec Services, AH and ESP</u>	240
<u>The Authentication Header (AH)</u>	241
<u>Encapsulated Security Payload (ESP)</u>	242
<u>IPsec modes</u>	243
<u>Tunnel mode</u>	243
<u>Transport mode</u>	243
<u>FreeS/WAN parts</u>	243
<u>KLIPS: Kernel IPsec Support</u>	243
<u>The Pluto daemon</u>	243
<u>The ipsec(8) command</u>	244
<u>Linux FreeS/WAN configuration file</u>	244
<u>Key management</u>	244
<u>Currently Implemented Methods</u>	244
<u>Methods not yet implemented</u>	245
<b><u>Mailing lists and newsgroups</u></b>	<b>247</b>
<u>Mailing lists about FreeS/WAN</u>	247
<u>The project mailing lists</u>	247
<u>Archives of the lists</u>	248
<u>Indexes of mailing lists</u>	248
<u>Lists for related software and topics</u>	248
<u>Products that include FreeS/WAN</u>	249
<u>Linux mailing lists</u>	249
<u>Lists for IETF working groups</u>	249
<u>Other mailing lists</u>	249
<u>Usenet newsgroups</u>	249
<b><u>Web links</u></b>	<b>250</b>
<u>The Linux FreeS/WAN Project</u>	250
<u>Add-ons and patches for FreeS/WAN</u>	250
<u>Distributions including FreeS/WAN</u>	251
<u>Things FreeS/WAN uses or could use</u>	251
<u>Other approaches to VPNs for Linux</u>	252
<u>The IPsec Protocols</u>	252
<u>General IPsec or VPN information</u>	252
<u>IPsec overview documents or slide sets</u>	252
<u>IPsec information in languages other than English</u>	252

# Table of Contents

## **Web links**

<u>RFCs and other reference documents</u> .....	252
<u>Analysis and critiques of IPsec protocols</u> .....	253
<u>Background information on IP</u> .....	253
<b><u>IPsec Implementations</u></b> .....	253
<u>Linux products</u> .....	253
<u>IPsec in router products</u> .....	253
<u>IPsec in firewall products</u> .....	254
<u>Operating systems with IPsec support</u> .....	254
<u>IPsec on network cards</u> .....	254
<u>Open source IPsec implementations</u> .....	254
<u>Interoperability</u> .....	255
<b><u>Linux links</u></b> .....	255
<u>Basic and tutorial Linux information</u> .....	255
<u>General Linux sites</u> .....	255
<u>Documentation</u> .....	256
<u>Advanced routing</u> .....	256
<u>Security for Linux</u> .....	256
<u>Linux firewalls</u> .....	256
<u>Miscellaneous Linux information</u> .....	257
<b><u>Crypto and security links</u></b> .....	257
<u>Crypto and security resources</u> .....	257
<u>Cryptography law and policy</u> .....	258
<u>Cryptography technical information</u> .....	258
<u>Computer and network security</u> .....	259
<u>Links to home pages</u> .....	260
 <b><u>Glossary for the Linux FreeS/WAN project</u></b> .....	 261
<u>Jump to a letter in the glossary</u> .....	261
<u>Other glossaries</u> .....	261
<u>Definitions</u> .....	262
 <b><u>Bibliography for the Linux FreeS/WAN project</u></b> .....	 292
 <b><u>IPsec RFCs and related documents</u></b> .....	 297
<u>The RFCs.tar.gz Distribution File</u> .....	297
<u>Other sources for RFCs &amp; Internet drafts</u> .....	297
<u>RFCs</u> .....	297
<u>Internet Drafts</u> .....	297
<u>FIPS standards</u> .....	297
<u>What's in the RFCs.tar.gz bundle?</u> .....	298
<u>Overview RFCs</u> .....	298
<u>Basic protocols</u> .....	298
<u>Key management</u> .....	298
<u>Details of various things used</u> .....	298
<u>Older RFCs which may be referenced</u> .....	298
<u>RFCs for secure DNS service, which IPsec may use</u> .....	298
<u>RFCs labelled "experimental"</u> .....	299

# Table of Contents

<b><u>IPsec RFCs and related documents</u></b>	
<u>Related RFCs</u> .....	299
<b><u>Distribution Roadmap: What's Where in Linux FreeS/WAN</u></b> .....	<b>300</b>
<u>Top directory</u> .....	300
<u>Documentation</u> .....	300
<u>KLIPS: kernel IP security</u> .....	300
<u>Pluto key and connection management daemon</u> .....	301
<u>Utils</u> .....	301
<u>Libraries</u> .....	302
<u>FreeS/WAN Library</u> .....	302
<u>Imported Libraries</u> .....	302
<b><u>User-Mode-Linux Testing guide</u></b> .....	<b>303</b>
<u>Preliminary Notes on BIND</u> .....	303
<u>Steps to Install UML for FreeS/WAN</u> .....	303
<b><u>Debugging the kernel with GDB</u></b> .....	<b>306</b>
<u>Other notes about debugging</u> .....	306
<b><u>User-Mode-Linux mysteries</u></b> .....	<b>307</b>
<b><u>Getting more info from uml netjig</u></b> .....	<b>308</b>
<b><u>How to configure to use "make check"</u></b> .....	<b>309</b>
<u>What is "make check"</u> .....	309
<u>Running "make check"</u> .....	309
<b><u>How to write a "make check" test</u></b> .....	<b>310</b>
<u>Structure of a test</u> .....	310
<u>The TESTLIST</u> .....	310
<u>Test kinds</u> .....	310
<u>Common parameters</u> .....	311
<u>KLIPStest paramaters</u> .....	312
<u>mkinsttest paramaters</u> .....	313
<u>rpm build install test paramaters</u> .....	314
<u>libtest paramaters</u> .....	314
<u>umlplutotest paramaters</u> .....	314
<u>umlXhost parameters</u> .....	314
<u>kernel patch test paramaters</u> .....	316
<u>module compile paramaters</u> .....	316
<b><u>Current pitfalls</u></b> .....	<b>318</b>
<b><u>Nightly regression testing</u></b> .....	<b>319</b>

# Table of Contents

<b><u>How to setup the nightly build</u></b> .....	<b>320</b>
<u>Files you need to know about</u> .....	320
<u>Configuring freeswan-regress-env.sh</u> .....	320

# Introduction

This section gives an overview of:

- what IP Security (IPsec) does
- how IPsec works
- why we are implementing it for Linux
- how this implementation works

This section is intended to cover only the essentials, *things you should know before trying to use FreeS/WAN*.

For more detailed background information, see the [history and politics](#) and [IPsec protocols](#) sections.

## IPsec, Security for the Internet Protocol

FreeS/WAN is a Linux implementation of the IPsec (IP security) protocols. IPsec provides [encryption](#) and [authentication](#) services at the IP (Internet Protocol) level of the network protocol stack.

Working at this level, IPsec can protect any traffic carried over IP, unlike other encryption which generally protects only a particular higher-level protocol — [PGP](#) for mail, [SSH](#) for remote login, [SSL](#) for web work, and so on. This approach has both considerable advantages and some limitations. For discussion, see our [IPsec section](#)

IPsec can be used on any machine which does IP networking. Dedicated IPsec gateway machines can be installed wherever required to protect traffic. IPsec can also run on routers, on firewall machines, on various application servers, and on end-user desktop or laptop machines.

Three protocols are used

- [AH](#) (Authentication Header) provides a packet-level authentication service
- [ESP](#) (Encapsulating Security Payload) provides encryption plus authentication
- [IKE](#) (Internet Key Exchange) negotiates connection parameters, including keys, for the other two

Our implementation has three main parts:

- [KLIPS](#) (kernel IPsec) implements AH, ESP, and packet handling within the kernel
- [Pluto](#) (an IKE daemon) implements IKE, negotiating connections with other systems
- various scripts provide an administrator's interface to the machinery

IPsec is optional for the current (version 4) Internet Protocol. FreeS/WAN adds IPsec to the Linux IPv4 network stack. Implementations of [IP version 6](#) are required to include IPsec. Work toward integrating FreeS/WAN into the Linux IPv6 stack has [started](#).

For more information on IPsec, see our [IPsec protocols](#) section, our collection of [IPsec links](#) or the [RFCs](#) which are the official definitions of these protocols.

## Interoperating with other IPsec implementations

IPsec is designed to let different implementations work together. We provide:

- a [list](#) of some other implementations
- information on [using FreeS/WAN with other implementations](#)

The VPN Consortium fosters cooperation among implementers and interoperability among implementations. Their [web site](#) has much more information.

## Advantages of IPsec

IPsec has a number of security advantages. Here are some independently written articles which discuss these:

[SANS institute papers](#). See the section on Encryption & VPNs.

[Cisco's white papers on "Networking Solutions"](#).

[Advantages of ISCS \(Linux Integrated Secure Communications System; includes FreeS/WAN and other software\)](#).

## Applications of IPsec

Because IPsec operates at the network layer, it is remarkably flexible and can be used to secure nearly any type of Internet traffic. Two applications, however, are extremely widespread:

- a [Virtual Private Network](#), or VPN, allows multiple sites to communicate securely over an insecure Internet by encrypting all communication between the sites.
- "Road Warriors" connect to the office from home, or perhaps from a hotel somewhere

There is enough opportunity in these applications that vendors are flocking to them. IPsec is being built into routers, into firewall products, and into major operating systems, primarily to support these applications. See our [list](#) of implementations for details.

We support both of those applications, and various less common IPsec applications as well, but we also add one of our own:

- opportunistic encryption, the ability to set up FreeS/WAN gateways so that any two of them can encrypt to each other, and will do so whenever packets pass between them.

This is an extension we are adding to the protocols. FreeS/WAN is the first prototype implementation, though we hope other IPsec implementations will adopt the technique once we demonstrate it. See [project goals](#) below for why we think this is important.

A somewhat more detailed description of each of these applications is below. Our [quickstart](#) section will show you how to build each of them.

## Using secure tunnels to create a VPN

A VPN, or **V**irtual **P**rivate **N**etwork lets two networks communicate securely when the only connection between them is over a third network which they do not trust.

## Introduction to FreeS/WAN

The method is to put a security gateway machine between each of the communicating networks and the untrusted network. The gateway machines encrypt packets entering the untrusted net and decrypt packets leaving it, creating a secure tunnel through it.

If the cryptography is strong, the implementation is careful, and the administration of the gateways is competent, then one can reasonably trust the security of the tunnel. The two networks then behave like a single large private network, some of whose links are encrypted tunnels through untrusted nets.

Actual VPNs are often more complex. One organisation may have fifty branch offices, plus some suppliers and clients, with whom it needs to communicate securely. Another might have 5,000 stores, or 50,000 point-of-sale devices. The untrusted network need not be the Internet. All the same issues arise on a corporate or institutional network whenever two departments want to communicate privately with each other.

Administratively, the nice thing about many VPN setups is that large parts of them are static. You know the IP addresses of most of the machines involved. More important, you know they will not change on you. This simplifies some of the admin work. For cases where the addresses do change, see the next section.

### Road Warriors

The prototypical "Road Warrior" is a traveller connecting to home base from a laptop machine. Administratively, most of the same problems arise for a telecommuter connecting from home to the office, especially if the telecommuter does not have a static IP address.

For purposes of this document:

- anyone with a dynamic IP address is a "Road Warrior".
- any machine doing IPsec processing is a "gateway". Think of the single-user road warrior machine as a gateway with a degenerate subnet (one machine, itself) behind it.

These require somewhat different setup than VPN gateways with static addresses and with client systems behind them, but are basically not problematic.

There are some difficulties which appear for some road warrior connections:

- Road Warriors who get their addresses via DHCP may have a problem. FreeS/WAN can quite happily build and use a tunnel to such an address, but when the DHCP lease expires, FreeS/WAN does not know that. The tunnel fails, and the only recovery method is to tear it down and re-build it.
- If Network Address Translation (NAT) is applied between the two IPsec Gateways, this breaks IPsec. IPsec authenticates packets on an end-to-end basis, to ensure they are not altered en route. NAT rewrites packets as they go by. See our firewalls document for details.

In most situations, however, FreeS/WAN supports road warrior connections just fine.

### Opportunistic encryption

One of the reasons we are working on FreeS/WAN is that it gives us the opportunity to add what we call opportunistic encryption. This means that any two FreeS/WAN gateways will be able to encrypt their traffic, even if the two gateway administrators have had no prior contact and neither system has any preset information about the other.

## Introduction to FreeS/WAN

Both systems pick up the authentication information they need from the DNS (domain name service), the service they already use to look up IP addresses. Of course the administrators must put that information in the DNS, and must set up their gateways with opportunistic encryption enabled. Once that is done, everything is automatic. The gateways look for opportunities to encrypt, and encrypt whatever they can. Whether they also accept unencrypted communication is a policy decision the administrator can make.

This technique can give two large payoffs:

- It reduces the administrative overhead for IPsec enormously. You configure your gateway and thereafter everything is automatic. The need to configure the system on a per-tunnel basis disappears. Of course, FreeS/WAN allows specifically configured tunnels to co-exist with opportunistic encryption, but we hope to make them unnecessary in most cases.
- It moves us toward a more secure Internet, allowing users to create an environment where message privacy is the default. All messages can be encrypted, provided the other end is willing to co-operate. See our history and politics of cryptography section for discussion of why we think this is needed.

Opportunistic encryption is not (yet?) a standard part of the IPsec protocols, but an extension we are proposing and demonstrating. For details of our design, see links below.

Only one current product we know of implements a form of opportunistic encryption. Secure sendmail will automatically encrypt server-to-server mail transfers whenever possible.

## The need to authenticate gateways

A complication, which applies to any type of connection — VPN, Road Warrior or opportunistic — is that a secure connection cannot be created magically. *There must be some mechanism which enables the gateways to reliably identify each other.* Without this, they cannot sensibly trust each other and cannot create a genuinely secure link.

Any link they do create without some form of authentication will be vulnerable to a man-in-the-middle attack. If Alice and Bob are the people creating the connection, a villain who can re-route or intercept the packets can pose as Alice while talking to Bob and pose as Bob while talking to Alice. Alice and Bob then both talk to the man in the middle, thinking they are talking to each other, and the villain gets everything sent on the bogus "secure" connection.

There are two ways to build links securely, both of which exclude the man-in-the middle:

- with **manual keying**, Alice and Bob share a secret key (which must be transmitted securely, perhaps in a note or via PGP or SSH) to encrypt their messages. For FreeS/WAN, such keys are stored in the ipsec.conf(5) file. Of course, if an enemy gets the key, all is lost.
- with **automatic keying**, the two systems authenticate each other and negotiate their own secret keys. The keys are automatically changed periodically.

Automatic keying is much more secure, since if an enemy gets one key only messages between the previous re-keying and the next are exposed. It is therefore the usual mode of operation for most IPsec deployment, and the mode we use in our setup examples. FreeS/WAN does support manual keying for special circumstances. See this section.

For automatic keying, the two systems must authenticate each other during the negotiations. There is a choice of methods for this:

- a *shared secret* provides authentication. If Alice and Bob are the only ones who know a secret and Alice receives a message which could not have been created without that secret, then Alice can safely believe the message came from Bob.
- a public key can also provide authentication. If Alice receives a message signed with Bob's private key (which of course only he should know) and she has a trustworthy copy of his public key (so that she can verify the signature), then she can safely believe the message came from Bob.

Public key techniques are much preferable, for reasons discussed later, and will be used in all our setup examples. FreeS/WAN does also support auto-keying with shared secret authentication. See this section.

## The FreeS/WAN project

For complete information on the project, see our web site, [freeswan.org](http://freeswan.org).

In summary, we are implementing the IPsec protocols for Linux and extending them to do opportunistic encryption.

### Project goals

Our overall goal in FreeS/WAN is to make the Internet more secure and more private.

Our IPsec implementation supports VPNs and Road Warriors of course. Those are important applications. Many users will want FreeS/WAN to build corporate VPNs or to provide secure remote access.

However, our goals in building it go beyond that. We are trying to help *build security into the fabric of the Internet* so that anyone who choses to communicate securely can do so, as easily as they can do anything else on the net.

More detailed objectives are:

- extend IPsec to do opportunistic encryption so that
  - ◆ any two systems can secure their communications without a pre-arranged connection
  - ◆ *secure connections can be the default*, falling back to unencrypted connections only if:
    - ◇ *both* the partner is not set up to co-operate on securing the connection
    - ◇ *and* your policy allows insecure connections
  - ◆ a significant fraction of all Internet traffic is encrypted
  - ◆ wholesale monitoring of the net (examples) becomes difficult or impossible
- help make IPsec widespread by providing an implementation with no restrictions:
  - ◆ freely available in source code under the GNU General Public License
  - ◆ running on a range of readily available hardware
  - ◆ not subject to US or other nations' export restrictions.  
Note that in order to avoid *even the appearance* of being subject to those laws, the project cannot accept software contributions — *not even one-line bug fixes* — from US residents or citizens.
- provide a high-quality IPsec implementation for Linux
  - ◆ portable to all CPUs Linux supports: (current list)
  - ◆ interoperable with other IPsec implementations: (current list)

If we can get opportunistic encryption implemented and widely deployed, then it becomes impossible for even huge well-funded agencies to monitor the net.

## Introduction to FreeS/WAN

See also our section on history and politics of cryptography, which includes our project leader's rationale for starting the project.

### Project team

Two of the team are from the US and can therefore contribute no code:

- John Gilmore: founder and policy-maker (home page)
- Hugh Daniel: project manager, Most Demented Tester, and occasionally Pointy-Haired Boss

The rest of the team are Canadians, working in Canada. (Why Canada?)

- Hugh Redelmeier: Pluto daemon programmer
- Richard Guy Briggs: KLIPS programmer
- Michael Richardson: hacker without portfolio
- Claudia Schmeing: documentation
- Sam Sgro: technical support via the mailing lists

The project is funded by civil libertarians who consider our goals worthwhile. Most of the team are paid for this work.

People outside this core team have made substantial contributions. See

- our CREDITS file
- the patches and add-ons section of our web references file
- lists below of user-written HowTos and other papers

Additional contributions are welcome. See the FAQ for details.

### Products containing FreeS/WAN

Unfortunately the export laws of some countries restrict the distribution of strong cryptography. FreeS/WAN is therefore not in the standard Linux kernel and not in all CD or web distributions.

FreeS/WAN is, however, quite widely used. Products we know of that use it are listed below. We would appreciate hearing, via the mailing lists, of any we don't know of.

### Full Linux distributions

FreeS/WAN is included in various general-purpose Linux distributions, mostly from countries (shown in brackets) with more sensible laws:

- SuSE Linux (Germany)
- Conectiva (Brazil)
- Mandrake (France)
- Debian
- the Polish(ed) Linux Distribution (Poland)
- Best Linux (Finland)

## Introduction to FreeS/WAN

For distributions which do not include FreeS/WAN and are not Redhat (which we develop and test on), there is additional information in our [compatibility](#) section.

The server edition of [Corel Linux](#) (Canada) also had FreeS/WAN, but Corel have dropped that product line.

## Linux kernel distributions

- [Working Overloaded Linux Kernel \(WOLK\)](#)

## Office server distributions

FreeS/WAN is also included in several distributions aimed at the market for turnkey business servers:

- [e-Smith](#) (Canada), which has recently been acquired and become the Network Server Solutions group of [Mitel Networks](#) (Canada)
- [ClarkConnect](#) from Point Clark Networks (Canada)
- [Trustix Secure Linux](#) (Norway)

## Firewall distributions

Several distributions intended for firewall and router applications include FreeS/WAN:

- The [Linux Router Project](#) produces a Linux distribution that will boot from a single floppy. The [LEAF](#) firewall project provides several different LRP-based firewall packages. At least one of them, Charles Steinkuehler's Dachstein, includes FreeS/WAN with X.509 patches.
- there are several distributions bootable directly from CD-ROM, usable on a machine without hard disk.
  - ◆ Dachstein (see above) can be used this way
  - ◆ [Gibraltar](#) is based on Debian GNU/Linux.
  - ◆ at time of writing, [Xiloo](#) is available only in Chinese. An English version is expected.
- [Astaro Security Linux](#) includes FreeS/WAN. It has some web-based tools for managing the firewall that include FreeS/WAN configuration management.
- [Linuxwall](#)
- [Smoothwall](#)
- [Devil Linux](#)
- Coyote Linux has a [Wolverine](#) firewall/VPN server

There are also several sets of scripts available for managing a firewall which is also acting as a FreeS/WAN IPsec gateway. See this [list](#).

## Firewall and VPN products

Several vendors use FreeS/WAN as the IPsec component of a turnkey firewall or VPN product.

Software-only products:

- [Linux Magic](#) offer a VPN/Firewall product using FreeS/WAN
- The Software Group's [Sentinet](#) product uses FreeS/WAN
- [Merilus](#) use FreeS/WAN in their Gateway Guardian firewall product

## Introduction to FreeS/WAN

Products that include the hardware:

- The [LASAT SafePipe\[tm\]](#) series. is an IPsec box based on an embedded MIPS running Linux with FreeS/WAN and a web-config front end. This company also host our freeswan.org web site.
- Merilus [Firecard](#) is a Linux firewall on a PCI card.
- [Kyzo](#) have a "pizza box" product line with various types of server, all running from flash. One of them is an IPsec/PPTP VPN server
- [PFN](#) use FreeS/WAN in some of their products

[Rebel.com](#), makers of the Netwinder Linux machines (ARM or Crusoe based), had a product that used FreeS/WAN. The company is in receivership so the future of the Netwinder is at best unclear. [PKIX patches](#) for FreeS/WAN developed at Rebel are listed in our web links document.

## Information sources

### This HowTo, in multiple formats

FreeS/WAN documentation up to version 1.5 was available only in HTML. Now we ship two formats:

- as HTML, one file for each doc section plus a global [Table of Contents](#)
- [one big HTML file](#) for easy searching

and provide a Makefile to generate other formats if required:

- [PDF](#)
- [Postscript](#)
- [ASCII text](#)

The Makefile assumes the htmldoc tool is available. You can download it from [Easy Software](#).

All formats should be available at the following websites:

- [FreeS/WAN project](#)
- [Linux Documentation Project](#)

The distribution tarball has only the two HTML formats.

**Note:** If you need the latest doc version, for example to see if anyone has managed to set up interoperation between FreeS/WAN and whatever, then you should download the current snapshot. What is on the web is documentation as of the last release. Snapshots have all changes I've checked in to date.

## RTFM (please Read The Fine Manuals)

As with most things on any Unix-like system, most parts of Linux FreeS/WAN are documented in online manual pages. We provide a list of [FreeS/WAN man pages](#), with links to HTML versions of them.

The man pages describing configuration files are:

- [ipsec.conf\(5\)](#)
- [ipsec.secrets\(5\)](#)

Man pages for common commands include:

- [ipsec\(8\)](#)
- [ipsec\\_pluto\(8\)](#)
- [ipsec\\_newhostkey\(8\)](#)
- [ipsec\\_auto\(8\)](#)

You can read these either in HTML using the links above or with the *man(1)* command.

In the event of disagreement between this HTML documentation and the man pages, the man pages are more likely correct since they are written by the implementers. Please report any such inconsistency on the [mailing list](#).

## Other documents in the distribution

Text files in the main distribution directory are README, INSTALL, CREDITS, CHANGES, BUGS and COPYING.

The Libdes encryption library we use has its own documentation. You can find it in the library directory..

## Background material

Throughout this documentation, I write as if the reader had at least a general familiarity with Linux, with Internet Protocol networking, and with the basic ideas of system and network security. Of course that will certainly not be true for all readers, and quite likely not even for a majority.

However, I must limit amount of detail on these topics in the main text. For one thing, I don't understand all the details of those topics myself. Even if I did, trying to explain everything here would produce extremely long and almost completely unreadable documentation.

If one or more of those areas is unknown territory for you, there are plenty of other resources you could look at:

### *Linux*

the [Linux Documentation Project](#) or a local [Linux User Group](#) and these [links](#)

### *IP networks*

Rusty Russell's [Networking Concepts HowTo](#) and these [links](#)

### *Security*

Schneier's book [Secrets and Lies](#) and these [links](#)

Also, I do make an effort to provide some background material in these documents. All the basic ideas behind IPsec and FreeS/WAN are explained here. Explanations that do not fit in the main text, or that not everyone will need, are often in the [glossary](#), which is the largest single file in this document set. There is also a [background](#) file containing various explanations too long to fit in glossary definitions. All files are heavily sprinkled with links to each other and to the glossary. *If some passage makes no sense to you, try the links.*

For other reference material, see the [bibliography](#) and our collection of [web links](#).

Of course, no doubt I get this (and other things) wrong sometimes. Feedback via the [mailing lists](#) is welcome.

## Archives of the project mailing list

Until quite recently, there was only one FreeS/WAN mailing list, and archives of it were:

- [Canada](#)
- [Holland](#)

The two archives use completely different search engines. You might want to try both.

More recently we have expanded to five lists, each with its own archive.

[More information](#) on mailing lists.

## User-written HowTo information

Various user-written HowTo documents are available. The ones covering FreeS/WAN-to-FreeS/WAN connections are:

- Jean-Francois Nadeau's [practical configurations](#) document
- Jens Zerbst's HowTo on [Using FreeS/WAN with dynamic IP addresses](#).
- an entry in Kurt Seifried's [Linux Security Knowledge Base](#).
- a section of David Ranch's [Trinity OS Guide](#)
- a section in David Bander's book [Linux Security Toolkit](#)

User-written HowTo material may be *especially helpful if you need to interoperate with another IPsec implementation*. We have neither the equipment nor the manpower to test such configurations. Users seem to be doing an admirable job of filling the gaps.

- list of user-written [interoperation HowTos](#) in our interop document

Check what version of FreeS/WAN user-written documents cover. The software is under active development and the current version may be significantly different from what an older document describes.

## Papers on FreeS/WAN

Two design documents show team thinking on new developments:

- [Opportunistic Encryption](#) by technical lead Henry Spencer and Pluto programmer Hugh Redelemeier
- discussion of [KLIPS redesign](#)

Both documents are works in progress and are frequently revised. For the latest version, see the [design mailing list](#). Comments should go to that list.

There is now an [Internet Draft on Opportunistic Encryption](#) by Michael Richardson, Hugh Redelmeier and Henry Spencer. This is a first step toward getting the protocol standardised so there can be multiple implementations of it. Discussion of it takes place on the [IETF IPsec Working Group](#) mailing list.

A number of papers giving further background on FreeS/WAN, or exploring its future or its applications, are also available:

## Introduction to FreeS/WAN

- Both Henry and Richard gave talks on FreeS/WAN at the 2000 [Ottawa Linux Symposium](#).
  - ♦ Richard's [slides](#)
  - ♦ Henry's paper
  - ♦ MP3 audio of their talks is available from the [conference page](#)
- *Moat: A Virtual Private Network Appliances and Services Platform* is a paper about large-scale (a few 100 links) use of FreeS/WAN in a production application at AT&T Research. It is available in Postscript or PDF from co-author Steve Bellovin's [papers list page](#).
- One of the Moat co-authors, John Denker, has also written
  - ♦ a [proposal](#) for how future versions of FreeS/WAN might interact with routing protocols
  - ♦ a [wishlist](#) of possible new features
- Bart Trojanowski's web page has a draft design for [hardware acceleration](#) of FreeS/WAN

Several of these provoked interesting discussions on the mailing lists, worth searching for in the [archives](#).

There are also several papers in languages other than English, see our [web links](#).

## License and copyright information

All code and documentation written for this project is distributed under either the GNU General Public License ([GPL](#)) or the GNU Library General Public License. For details see the COPYING file in the distribution.

Not all code in the distribution is ours, however. See the CREDITS file for details. In particular, note that the [Libdes](#) library and the version of [MD5](#) that we use each have their own license.

## Distribution sites

FreeS/WAN is available from a number of sites.

### Primary site

Our primary site, is at xs4all (Thanks, folks!) in Holland:

- [HTTP](#)
- [FTP](#)

### Mirrors

There are also mirror sites all over the world:

- [Eastern Canada](#) (limited resouces)
- [Eastern Canada](#) (has older versions too)
- [Eastern Canada](#) (has older versions too)
- [Japan](#)
- [Hong Kong](#)
- [Denmark](#)
- [the UK](#)
- [Slovak Republic](#)
- [Australia](#)

- [technolust](#)
- [Germany](#)
- Ivan Moore's [site](#)
- the [Crypto Archive](#) on the [Security Portal](#) site
- [Wiretapped.net](#) in Australia

Thanks to those folks as well.

## The "munitions" archive of Linux crypto software

There is also an archive of Linux crypto software called "munitions", with its own mirrors in a number of countries. It includes FreeS/WAN, though not always the latest version. Some of its sites are:

- [Germany](#)
- [Italy](#)
- [Netherlands](#)

Any of those will have a list of other "munitions" mirrors. There is also a CD available.

## Links to other sections

For more detailed background information, see:

- [history and politics](#) of cryptography
- [IPsec protocols](#)

To begin working with FreeS/WAN, go to our [quickstart](#) guide.

# Upgrading to FreeS/WAN 2.x

## New! Built in Opportunistic connections

Out of the box, FreeS/WAN 2.x will attempt to encrypt all your IP traffic. It will try to establish IPsec connections for:

- IP traffic from the Linux box on which you have installed FreeS/WAN, and
- outbound IP traffic routed through that Linux box (eg. from a protected subnet).

FreeS/WAN 2.x uses *hidden, automatically enabled ipsec.conf connections* to do this.

This behaviour is part of our campaign to get Opportunistic Encryption (OE) widespread in the Linux world, so that any two Linux boxes can encrypt to one another without prearrangement. There's one catch, however: you must set up a few DNS records to distribute RSA public keys and (if applicable) IPsec gateway information.

If you start FreeS/WAN before you have set up these DNS records, your connectivity will be slow, and messages relating to the built in connections will clutter your logs. If you are unable to set up DNS for OE, you will wish to disable the hidden connections.

## Upgrading Opportunistic Encryption to 2.01 (or later)

As of FreeS/WAN 2.01, Opportunistic Encryption (OE) uses DNS TXT resource records (RRs) only (rather than TXT with KEY). This change causes a "flag day". Users of FreeS/WAN 2.00 (or earlier) OE who are upgrading may need to post additional resource records.

If you are running initiate-only OE, you *must* put up a TXT record in any forward domain as per our quickstart instructions. This replaces your old forward KEY.

If you are running full OE, you require no updates. You already have the needed TXT record in the reverse domain. However, to facilitate future features, you may also wish to publish that TXT record in a forward domain as instructed here.

If you are running OE on a gateway (and encrypting on behalf of subnetted boxes) you require no updates. You already have the required TXT record in your gateway's reverse map, and the TXT records for any subnetted boxes require no updating. However, to facilitate future features, you may wish to publish your gateway's TXT record in a forward domain as shown here.

During the transition, you may wish to leave any old KEY records up for some time. They will provide limited backward compatibility.

## New! Policy Groups

We want to make it easy for you to declare security policy as it applies to IPsec connections.

Policy Groups make it simple to say:

## Introduction to FreeS/WAN

- These are the folks I want to talk to in the clear.
- These spammers' domains — I don't want to talk to them at all.
- To talk to the finance department, I must use IPsec.
- For any other communication, try to encrypt, but it's okay if we can't.

FreeS/WAN then implements these policies, creating OE connections if and when needed. You can use Policy Groups along with connections you explicitly define in `ipsec.conf`.

For more information, see our [Policy Group HOWTO](#).

## New! Packetdefault Connection

Free/SWAN 2.x ships with the *automatically enabled, hidden connection packetdefault*. This configures a FreeS/WAN box as an OE gateway for any hosts located behind it. As mentioned above, you must configure some [DNS records](#) for OE to work.

As the name implies, this connection functions as a default. If you have more specific connections, such as policy groups which configure your FreeS/WAN box as an OE gateway for a local subnet, these will apply before *packetdefault*. You can view *packetdefault*'s specifics in [man ipsec.conf](#).

## FreeS/WAN now disables Reverse Path Filtering

FreeS/WAN often doesn't work with reverse path filtering. At start time, FreeS/WAN now turns `rp_filter` off, and logs a warning.

FreeS/WAN does not turn it back on again. You can do this yourself with a command like:

```
echo 1 > /proc/sys/net/ipv4/conf/eth0/rp_filter
```

For `eth0`, substitute the interface which FreeS/WAN was affecting.

## Revised *ipsec.conf*

### No promise of compatibility

The FreeS/WAN team promised config-file compatibility throughout the 1.x series. That means a 1.5 config file can be directly imported into a fresh 1.99 install with no problems.

With FreeS/WAN 2.x, we've given ourselves permission to make the config file easier to use. The cost: some FreeS/WAN 1.x configurations will not work properly. Many of the new features are, however, backward compatible.

### Most *ipsec.conf* files will work fine

... so long as you paste this line, *with no preceding whitespace*, at the top of your config file:

```
version 2
```

## Backward compatibility patch

If the new defaults bite you, use [this \*ipsec.conf\* fragment](#) to simulate the old default values.

### Details

We've obsoleted various directives which almost no one was using:

```
dump
plutobackgroundload
no_eroute_pass
lifetime
rekeystart
rekeytries
```

For most of these, there is some other way to elicit the desired behaviour. See [this post](#).

We've made some settings, which almost everyone was using, defaults. For example:

```
interfaces=%defaultroute
plutoload=%search
plutostart=%search
uniqueids=yes
```

We've also changed some default values to help with OE and Policy Groups:

```
authby=rsasig    ## not secret!!!
lefttrsasigkey=%dnsondemand ## looks up missing keys in DNS when needed.
righttrsasigkey=%dnsondemand
```

Of course, you can still override any defaults by explicitly declaring something else in your connection.

[A post with a list of many \*ipsec.conf\* changes.](#)

[Current \*ipsec.conf\* manual.](#)

## Upgrading from 1.x RPMs to 2.x RPMs

Note: When upgrading from 1-series to 2-series RPMs, *rpm -U* will not work.

You must instead erase the 1.x RPMs, then install the 2.x set:

```
rpm -e freeswan

rpm -e freeswan-module
```

On erasing, your old *ipsec.conf* should be moved to *ipsec.conf.rpmsave*. Keep this. You will probably want to copy your existing connections to the end of your new 2.x file.

Install the RPMs suitable for your kernel version, such as:

```
rpm -ivh freeswan-module-2.03_2.4.20_20.9-0.i386.rpm
```

## Introduction to FreeS/WAN

```
rpm -ivh freeswan-userland-2.03_2.4.20_20.9-0.i386.rpm
```

Or, to splice the files:

```
cat /etc/ipsec.conf /etc/ipsec.conf.rpmsave > /etc/ipsec.conf.tmp  
mv /etc/ipsec.conf.tmp /etc/ipsec.conf
```

Then, remove the redundant *conn %default* and *config setup* sections. Unless you have done any special configuring here, you'll likely want to remove the 1.x versions. Remove *conn OEsself*, if present.

# Quickstart Guide to Opportunistic Encryption

## Purpose

This page will get you started using Linux FreeS/WAN with opportunistic encryption (OE). OE enables you to set up IPsec tunnels without co-ordinating with another site administrator, and without hand configuring each tunnel. If enough sites support OE, a "FAX effect" occurs, and many of us can communicate without eavesdroppers.

## OE "flag day"

As of FreeS/WAN 2.01, OE uses DNS TXT resource records (RRs) only (rather than TXT with KEY). This change causes a "flag day". Users of FreeS/WAN 2.00 (or earlier) OE who are upgrading may require additional resource records, as detailed in our upgrading document. OE setup instructions here are for 2.02 or later.

## Requirements

To set up opportunistic encryption, you will need:

- a Linux box. For OE to the public Internet, this box must NOT be behind Network Address Translation (NAT).
- to install Linux FreeS/WAN 2.02 or later
- either control over your reverse DNS (for full opportunism) or the ability to write to some forward domain (for initiator-only). This free DNS service explicitly supports forward TXT records for FreeS/WAN use.
- (for full opportunism) a static IP

Note: Currently, only Linux FreeS/WAN supports opportunistic encryption.

## RPM install

Our instructions are for a recent Red Hat with a 2.4-series stock or Red Hat updated kernel. For other ways to install, see our install document.

## Download RPMs

If we have prebuilt RPMs for your Red Hat system, this command will get them:

```
ncftpget ftp://ftp.xs4all.nl/pub/crypto/freeswan/binaries/RedHat-RPMs/`uname -r` | tr -d 'a-wy'
```

If that fails, you will need to try another install method. Our kernel modules **will only work on the Red Hat kernel they were built for**, since they are very sensitive to small changes in the kernel.

If it succeeds, you will have userland tools, a kernel module, and an RPM signing key:

```
freeswan-module-2.03_2.4.20_20.9-0.i386.rpm
```

## Introduction to FreeS/WAN

```
freeswan-userland-2.03_2.4.20_20.9-0.i386.rpm  
freeswan-rpmsign.asc
```

### Check signatures

If you're running RedHat 8.x or later, import the RPM signing key into the RPM database:

```
rpm --import freeswan-rpmsign.asc
```

For RedHat 7.x systems, you'll need to add it to your PGP keyring:

```
pgp -ka freeswan-rpmsign.asc
```

Check the digital signatures on both RPMs using:

```
rpm --checksig freeswan*.rpm
```

You should see that these signatures are good:

```
freeswan-module-2.03_2.4.20_20.9-0.i386.rpm: pgp md5 OK  
freeswan-userland-2.03_2.4.20_20.9-0.i386.rpm: pgp md5 OK
```

### Install the RPMs

Become root:

```
su
```

Install your RPMs with:

```
rpm -ivh freeswan*.rpm
```

If you're upgrading from FreeS/WAN 1.x RPMs, and have problems with that command, see [this note](#).

Then, start FreeS/WAN:

```
service ipsec start
```

### Test

To check that you have a successful install, run:

```
ipsec verify
```

You should see as part of the *verify* output:

```
Checking your system to see if IPsec got installed and started correctly  
Version check and ipsec on-path [OK]  
Checking for KLIPS support in kernel [OK]  
Checking for RSA private key (/etc/ipsec.secrets) [OK]  
Checking that pluto is running [OK]  
...
```

If any of these first four checks fails, see our [troubleshooting guide](#).

## Our Opportunistic Setups

### Full or partial opportunism?

Determine the best form of opportunism your system can support.

- For [full opportunism](#), you'll need a static IP and either control over your reverse DNS or an ISP that can add the required TXT record for you.
- If you have a dynamic IP, and/or write access to forward DNS only, you can do [initiate-only opportunism](#)
- To protect traffic bound for real IPs behind your gateway, use [this form of full opportunism](#).

### Initiate-only setup

#### Restrictions

When you set up initiate-only Opportunistic Encryption (iOE):

- there will be *no incoming connection requests*; you can initiate all the IPsec connections you need.
- *only one machine is visible* on your end of the connection.
- iOE also protects traffic on behalf of [NATted](#) hosts behind the iOE box.

You cannot network a group of initiator-only machines if none of these is capable of responding to OE. If one is capable of responding, you may be able to create a hub topology using routing.

### Create and publish a forward DNS record

#### Find a domain you can use

Find a DNS forward domain (e.g. example.com) where you can publish your key. You'll need access to the DNS zone files for that domain. This is common for a domain you own. Some free DNS providers, such as [this one](#), also provide this service.

Dynamic IP users take note: the domain where you place your key need not be associated with the IP address for your system, or even with your system's usual hostname.

#### Choose your ID

Choose a name within that domain which you will use to identify your machine. It's convenient if this can be the same as your hostname:

```
[root@xy root]# hostname --fqdn  
xy.example.com
```

This name in FQDN (fully-qualified domain name) format will be your ID, for DNS key lookup and IPsec negotiation.

### Create a forward TXT record

Generate a forward TXT record containing your system's public key with a command like:

```
ipsec showhostkey --txt @xy.example.com
```

using your chosen ID in place of xy.example.com. This command takes the contents of /etc/ipsec.secrets and reformats it into something usable by ISC's BIND. The result should look like this (with the key data trimmed down for clarity):

```
; RSA 2192 bits    xy.example.com    Thu Jan  2 12:41:44 2003
IN      TXT        "X-IPsec-Server(10)=@xy.example.com"
"AQOF8tZ2... ..+buFuFn/"
```

### Publish the forward TXT record

Insert the record into DNS, or have a system administrator do it for you. It may take up to 48 hours for the record to propagate, but it's usually much quicker.

### Test that your key has been published

Check your DNS work

```
ipsec verify --host xy.example.com
```

As part of the *verify* output, you ought to see something like:

```
...
Looking for TXT in forward map: xy.example.com      [OK]
...
```

For this type of opportunism, only the forward test is relevant; you can ignore the tests designed to find reverse records.

### Configure, if necessary

If your ID is the same as your hostname, you're ready to go. FreeS/WAN will use its built-in connections to create your iOE functionality.

If you have chosen a different ID, you must tell FreeS/WAN about it via ipsec.conf:

```
config setup
    myid=@myname.freedns.example.com
```

and restart FreeS/WAN:

```
service ipsec restart
```

The new ID will be applied to the built-in connections.

Note: you can create more complex iOE configurations as explained in our policy groups document, or disable OE using these instructions.

## Test

That's it! Test your connections.

## Full Opportunism

Full opportunism allows you to initiate and receive opportunistic connections on your machine.

### Put a TXT record in a Forward Domain

To set up full opportunism, first set up a forward TXT record as for initiator-only OE, using an ID (for example, your hostname) that resolves to your IP. Do not configure */etc/ipsec.conf*, but continue with the instructions for full opportunism, below.

Note that this forward record is not currently necessary for full OE, but will facilitate future features.

### Put a TXT record in Reverse DNS

You must be able to publish your DNS RR directly in the reverse domain. FreeS/WAN will not follow a PTR which appears in the reverse, since a second lookup at connection start time is too costly.

#### Create a Reverse DNS TXT record

This record serves to publicize your FreeS/WAN public key. In addition, it lets others know that this machine can receive opportunistic connections, and asserts that the machine is authorized to encrypt on its own behalf.

Use the command:

```
ipsec showhostkey --txt 192.0.2.11
```

where you replace 192.0.2.11 with your public IP.

The record (with key shortened) looks like:

```
; RSA 2048 bits xy.example.com Sat Apr 15 13:53:22 2000
IN TXT "X-IPsec-Server(10)=192.0.2.11" " AQOF8tZ2...+buFuFn/ "
```

#### Publish your TXT record

Send these records to your ISP, to be published in your IP's reverse map. It may take up to 48 hours for these to propagate, but usually takes much less time.

## Test your DNS record

Check your DNS work with

```
ipsec verify --host xy.example.com
```

As part of the *verify* output, you ought to see something like:

```
...  
Looking for TXT in reverse map: 11.2.0.192.in-addr.arpa [OK]  
...
```

which indicates that you've passed the reverse-map test.

## No Configuration Needed

FreeS/WAN 2.x ships with full OE enabled, so you don't need to configure anything. To enable OE out of the box, FreeS/WAN 2.x uses the policy group *private-or-clear*, which creates IPsec connections if possible (using OE if needed), and allows traffic in the clear otherwise. You can create more complex OE configurations as described in our [policy groups document](#), or disable OE using [these instructions](#).

If you've previously configured for initiator-only opportunism, remove *myid=* from *config setup*, so that peer FreeS/WANs will look up your key by IP. Restart FreeS/WAN so that your change will take effect, with

```
service ipsec restart
```

## Consider Firewalling

If you are running a default install of RedHat 8.x, take note: you will need to alter your iptables rule setup to allow IPsec traffic through your firewall. See [our firewall document](#) for sample *iptables* rules.

## Test

That's it. Now, [test your connection](#).

## Test

Instructions are in the next section.

## Testing opportunistic connections

Be sure IPsec is running. You can see whether it is with:

```
ipsec setup status
```

If need be, you can restart it with:

```
service ipsec restart
```

Load a FreeS/WAN test website from the host on which you're running FreeS/WAN. Note: the feds may be watching these sites. Type one of:

```
links oetest.freeswan.org
```

```
links oetest.freeswan.nl
```

A positive result looks like this:

## Introduction to FreeS/WAN

You seem to be connecting from: 192.0.2.11 which DNS says is:  
gateway.example.com

---

```
Status E-route
OE      enabled    16    192.139.46.73/32    ->    192.0.2.11/32    =>
tun0x2097@192.0.2.11
OE      enabled    176   192.139.46.77/32    ->    192.0.2.11/32    =>
tun0x208a@192.0.2.11
```

If you see this, congratulations! Your OE host or gateway will now encrypt its own traffic whenever it can. For more OE tests, please see our [testing document](#). If you have difficulty, see our [OE troubleshooting tips](#).

## Now what?

Please see our [policy groups document](#) for more ways to set up Opportunistic Encryption.

You may also wish to make some [pre-configured connections](#).

## Notes

- We assume some facts about your system in order to make Opportunistic Encryption easier to configure. For example, we assume that you wish to have FreeS/WAN secure your default interface.
- You may change this, and other settings, by altering the *config setup* section in */etc/ipsec.conf*.
- Note that the built-in connections used to build policy groups do not inherit from *conn default*.
- If you fail to define your local identity and do not fill in your reverse DNS entry, you will not be able to use OE.

## Troubleshooting OE

See the OE troubleshooting hints in our [troubleshooting guide](#).

## Known Issues

Please see [this list](#) of known issues with Opportunistic Encryption.

# How to Configure Linux FreeS/WAN with Policy Groups

## What are Policy Groups?

**Policy Groups** are an elegant general mechanism to configure FreeS/WAN. They are useful for many FreeS/WAN users.

In previous FreeS/WAN versions, you needed to configure each IPsec connection explicitly, on both local and remote hosts. This could become complex.

By contrast, Policy Groups allow you to set local IPsec policy for lists of remote hosts and networks, simply by listing the hosts and networks which you wish to have special treatment in one of several Policy Group files. FreeS/WAN then internally creates the connections needed to implement each policy.

In the next section we describe our five Base Policy Groups, which you can use to configure IPsec in many useful ways. Later, we will show you how to create an IPsec VPN using one line of configuration for each remote host or network.

## Built-In Security Options

FreeS/WAN offers these Base Policy Groups:

### *private*

FreeS/WAN only communicates privately with the listed CIDR blocks. If needed, FreeS/WAN attempts to create a connection opportunistically. If this fails, FreeS/WAN blocks communication. Inbound blocking is assumed to be done by the firewall. FreeS/WAN offers firewall hooks but no modern firewall rules to help with inbound blocking.

### *private-or-clear*

FreeS/WAN prefers private communication with the listed CIDR blocks. If needed, FreeS/WAN attempts to create a connection opportunistically. If this fails, FreeS/WAN allows traffic in the clear.

### *clear-or-private*

FreeS/WAN communicates cleartext with the listed CIDR blocks, but also accepts inbound OE connection requests from them. Also known as passive OE (pOE), this policy may be used to create an opportunistic responder.

### *clear*

FreeS/WAN only communicates cleartext with the listed CIDR blocks.

### *block*

FreeS/WAN blocks traffic to and from the listed CIDR blocks. Inbound blocking is assumed to be done by the firewall. FreeS/WAN offers firewall hooks but no modern firewall rules to help with inbound blocking.

Notes:

- Base Policy Groups apply to communication with this host only.

- The most specific rule (whether policy or pre-configured connection) applies. This has several practical applications:
  - ◆ If CIDR blocks overlap, FreeS/WAN chooses the most specific applicable block.
  - ◆ This decision also takes into account any pre-configured connections you may have.
  - ◆ If the most specific connection is a pre-configured connection, the following procedure applies. If that connection is up, it will be used. If it is routed, it will be brought up. If it is added, no action will be taken.
- Base Policy Groups are created using built-in connections. Details in [man ipsec.conf](#).
- All Policy Groups are bidirectional. [This chart](#) shows some technical details. FreeS/WAN does not support one-way encryption, since it can give users a false sense of security.

## Using Policy Groups

The Base Policy Groups which build IPsec connections rely on Opportunistic Encryption. To use the following examples, you must first become OE-capable, as described in our [quickstart guide](#).

### Example 1: Using a Base Policy Group

Simply place CIDR blocks ([names](#), IPs or IP ranges) in `/etc/ipsec.d/policies/[groupname]`, and reread the policy group files.

For example, the *private-or-clear* policy tells FreeS/WAN to prefer encrypted communication to the listed CIDR blocks. Failing that, it allows talk in the clear.

To make this your default policy, place [fullnet](#) in the *private-or-clear* policy group file:

```
[root@xy root]# cat /etc/ipsec.d/policies/private-or-clear
# This file defines the set of CIDRs (network/mask-length) to which
# communication should be private, if possible, but in the clear otherwise.
....
0.0.0.0/0
```

and reload your policies with

```
ipsec auto --rereadgroups
```

Use [this test](#) to verify opportunistic connections.

### Example 2: Defining IPsec Security Policy with Groups

Defining IPsec security policy with Base Policy Groups is like creating a shopping list: just put CIDR blocks in the appropriate group files. For example:

```
[root@xy root]# cd /etc/ipsec.d/policies
[root@xy policies]# cat private
192.0.2.96/27          # The finance department
192.0.2.192/29        # HR
192.0.2.12            # HR gateway
irc.private.example.com # Private IRC server

[root@xy policies]# cat private-or-clear
```

## Introduction to FreeS/WAN

```
0.0.0.0/0                                # My default policy: try to encrypt.

[root@xy policies]# cat clear
192.0.2.18/32                           # My POP3 server
192.0.2.19/32                           # My Web proxy

[root@xy policies]# cat block
spamsource.example.com
```

To make these settings take effect, type:

```
ipsec auto --rereadgroups
```

Notes:

- For opportunistic connection attempts to succeed, all participating FreeS/WAN hosts and gateways must be configured for OE.
- Examples 3 through 5 show how to implement a detailed *private* policy.
- **Warning:** Using DNS names in policy files and ipsec.conf can be tricky. If the name does not resolve, the policy will not be implemented for that name. It is therefore safer either to use IPs, or to put any critical names in /etc/hosts. We plan to implement periodic DNS retry to help with this. Names are resolved at FreeS/WAN startup, or when the policies are reloaded. Unfortunately, name lookup can hold up the startup process. If you have fast DNS servers, the problem may be less severe.

### Example 3: Creating a Simple IPsec VPN with the *private* Group

You can create an IPsec VPN between several hosts, with only one line of configuration per host, using the *private* policy group.

First, use our [quickstart guide](#) to set up each participating host with a FreeS/WAN install and OE.

In one host's `/etc/ipsec.d/policies/private`, list the peers to which you wish to protect traffic. For example:

```
[root@xy root]# cd /etc/ipsec.d/policies
[root@xy policies]# cat private
192.0.2.9                                # several hosts at example.com
192.0.2.11
192.0.2.12
irc.private.example.com
```

Copy the *private* file to each host. Remove the local host, and add the initial host.

```
scp2 /etc/ipsec.d/policies/private root@192.0.2.12:/etc/ipsec.d/policies/private
```

On each host, reread the policy groups with

```
ipsec auto --rereadgroups
```

That's it! You're configured.

Test by pinging between two hosts. After a second or two, traffic should flow, and

```
ipsec eroute
```

should yield something like

```
192.0.2.11/32    -> 192.0.2.8/32    => tun0x149f@192.0.2.8
```

where your host IPs are substituted for 192.0.2.11 and 192.0.2.8.

If traffic does not flow, there may be an error in your OE setup. Revisit our [quickstart guide](#).

Our next two examples show you how to add subnets to this IPsec VPN.

### Example 4: New Policy Groups to Protect a Subnet

To protect traffic to a subnet behind your FreeS/WAN gateway, you'll need additional DNS records, and new policy groups. To set up the DNS, see our [quickstart guide](#). To create five new policy groups for your subnet, copy these connections to `/etc/ipsec.conf`. Substitute your subnet's IPs for 192.0.2.128/29.

```
conn private-net
    also=private # inherits settings (eg. auto=start) from built in conn
    leftsubnet=192.0.2.128/29 # your subnet's IPs here

conn private-or-clear-net
    also=private-or-clear
    leftsubnet=192.0.2.128/29

conn clear-or-private-net
    also=clear-or-private
    leftsubnet=192.0.2.128/29

conn clear-net
    also=clear
    leftsubnet=192.0.2.128/29

conn block-net
    also=block
    leftsubnet=192.0.2.128/29
```

Copy the gateway's files to serve as the initial policy group files for the new groups:

```
cp -p /etc/ipsec.d/policies/private /etc/ipsec.d/policies/private-net
cp -p /etc/ipsec.d/policies/private-or-clear /etc/ipsec.d/policies/private-or-clear-net
cp -p /etc/ipsec.d/policies/clear-or-private /etc/ipsec.d/policies/clear-or-private-net
cp -p /etc/ipsec.d/policies/clear /etc/ipsec.d/policies/clear-net
cp -p /etc/ipsec.d/policies/block /etc/ipsec.d/policies/block
```

**Tip:** *Since a missing policy group file is equivalent to a file with no entries, you need only create files for the connections you'll use.*

To test one of your new groups, place the fullnet 0.0.0.0/0 in `private-or-clear-net`. Perform the subnet test in [our quickstart guide](#). You should see a connection, and

```
ipsec eroute
```

should include an entry which mentions the subnet node's IP and the OE test site IP, like this:

192.0.2.131/32 -> 192.139.46.77/32 => tun0x149f@192.0.2.11

### Example 5: Adding a Subnet to the VPN

Suppose you wish to secure traffic to a subnet 192.0.2.192/29 behind a FreeS/WAN box 192.0.2.12.

First, add DNS entries to configure 192.0.2.12 as an opportunistic gateway for that subnet. Instructions are in our [quickstart guide](#). Next, create a *private-net* group on 192.0.2.12 as described in [Example 4](#).

On each other host, add the subnet 192.0.2.192/29 to *private*, yielding for example

```
[root@xy root]# cd /etc/ipsec.d/policies
[root@xy policies]# cat private
192.0.2.9           # several hosts at example.com
192.0.2.11
192.0.2.12          # HR department gateway
192.0.2.192/29      # HR subnet
irc.private.example.com
```

and reread policy groups with

```
ipsec auto --rereadgroups
```

That's all the configuration you need.

Test your VPN by pinging from a machine on 192.0.2.192/29 to any other host:

```
[root@192.0.2.194]# ping 192.0.2.11
```

After a second or two, traffic should flow, and

```
ipsec eroute
```

should yield something like

192.0.2.11/32 -> 192.0.2.194/32 => tun0x149f@192.0.2.12

Key:

1. 192.0.2.11/32 Local start point of the protected traffic.
2. 192.0.2.194/32 Remote end point of the protected traffic.
3. 192.0.2.12 Remote FreeS/WAN node (gateway or host). May be the same as (2).
4. [not shown] Local FreeS/WAN node (gateway or host), where you've produced the output. May be the same as (1).

For additional assurance, you can verify with a packet sniffer that the traffic is being encrypted.

Note

- Because strangers may also connect via OE, this type of VPN may require a stricter firewalling policy than a conventional VPN.

## Appendix

### Our Hidden Connections

Our Base Policy Groups are created using hidden connections. These are spelled out in [man ipsec.conf](#) and defined in `/usr/local/lib/ipsec/_confread`.

### Custom Policy Groups

A policy group is built using a special connection description in `ipsec.conf`, which:

- is **generic**. It uses `right=[%group]%opportunisticgroup` rather than specific IPs. The connection is cloned for every name or IP range listed in its Policy Group file.
- often has a **failure rule**. This rule, written `failureshunt=[passthrough|drop|reject|none]`, tells FreeS/WAN what to do with packets for these CIDRs if it fails to establish the connection. Default is `none`.

To create a new group:

1. Create its connection definition in `ipsec.conf`.
2. Create a Policy Group file in `/etc/ipsec.d/policies` with the same name as your connection.
3. Put a CIDR block in that file.
4. Reread groups with `ipsec auto --rereadgroups`.
5. Test: `ping` to activate any OE connection, and view results with `ipsec eroute`.

### Disabling Opportunistic Encryption

To disable OE (eg. policy groups and packetdefault), cut and paste the following lines to `/etc/ipsec.conf`:

```
conn block
    auto=ignore

conn private
    auto=ignore

conn private-or-clear
    auto=ignore

conn clear-or-private
    auto=ignore

conn clear
    auto=ignore

conn packetdefault
    auto=ignore
```

Restart FreeS/WAN so that the changes take effect:

```
ipsec setup restart
```

# FreeS/WAN FAQ

This is a collection of questions and answers, mostly taken from the FreeS/WAN [mailing list](#). See the project [web site](#) for more information. All the FreeS/WAN documentation is online there.

Contributions to the FAQ are welcome. Please send them to the project [mailing list](#).

---

## Index of FAQ questions

- [What is FreeS/WAN?](#)
- [How do I report a problem or seek help?](#)
- [Can I get ...](#)
  - ◆ [... an off-the-shelf system that includes FreeS/WAN?](#)
  - ◆ [... contractors or staff who know FreeS/WAN?](#)
  - ◆ [... commercial support?](#)
- [Release questions](#)
  - ◆ [What is the current release?](#)
  - ◆ [When is the next release?](#)
  - ◆ [Are there known bugs in the current release?](#)
- [Modifications and contributions](#)
  - ◆ [Can I modify FreeS/WAN to ...?](#)
  - ◆ [Can I contribute to the project?](#)
  - ◆ [Is there detailed design documentation?](#)
- [Will FreeS/WAN work in my environment?](#)
  - ◆ [Can FreeS/WAN talk to ... ?](#)
  - ◆ [Can different FreeS/WAN versions talk to each other?](#)
  - ◆ [Is there a limit on throughput?](#)
  - ◆ [Is there a limit on number of connections?](#)
  - ◆ [Is a ... fast enough to handle FreeS/WAN with my loads?](#)
- [Will FreeS/WAN work on ...](#)
  - ◆ [... my version of Linux?](#)
  - ◆ [... non-Intel CPUs?](#)
  - ◆ [... multiprocessors?](#)
  - ◆ [... an older kernel?](#)
  - ◆ [... the latest kernel version?](#)
  - ◆ [... unusual network hardware?](#)
  - ◆ [... a VLAN \(802.1q\) network?](#)
- [Does FreeS/WAN support ...](#)
  - ◆ [... site-to-site VPN applications](#)
  - ◆ [... remote users connecting to a LAN](#)
  - ◆ [... remote users using shared secret authentication?](#)
  - ◆ [... wireless networks](#)
  - ◆ [... X.509 or other PKI certificates?](#)
  - ◆ [... user authentication \(Radius, SecureID, Smart Card ...\)?](#)
  - ◆ [... NAT traversal](#)
  - ◆ [... assigning a "virtual identity" to a remote system?](#)
  - ◆ [... single DES encryption?](#)
  - ◆ [... AES encryption?](#)
  - ◆ [... other encryption algorithms?](#)

## Introduction to FreeS/WAN

- Can I...
  - ◆ ...use policy groups along with explicitly configured connections?
  - ◆ ...turn off policy groups?
  - ◆ ... reload connection info without restarting?
  - ◆ ... use several masqueraded subnets?
  - ◆ ... use subnets masqueraded to the same addresses?
  - ◆ ... assign a road warrior an address on my net (a virtual identity)?
  - ◆ ... support many road warriors with one gateway?
  - ◆ ... have many road warriors using shared secret authentication?
  - ◆ ... use Quality of Service routing with FreeS/WAN?
  - ◆ ... recognise dead tunnels and shut them down?
  - ◆ ... build IPsec tunnels over a demand-dialed link?
  - ◆ ... build GRE, L2TP or PPTP tunnels over IPsec?
  - ◆ ... use Network Neighborhood (Samba, NetBIOS) over IPsec?
- Life's little mysteries
  - ◆ I cannot ping ....
  - ◆ It takes forever to ...
  - ◆ I send packets to the tunnel with route(8) but they vanish
  - ◆ When a tunnel goes down, packets vanish
  - ◆ The firewall ate my packets!
  - ◆ Dropped connections
  - ◆ Disappearing %defaultroute
  - ◆ TCPdump on the gateway shows strange things
  - ◆ Traceroute does not show anything between the gateways
- Testing in stages (or .... works but ... doesn't)
  - ◆ Manually keyed connections don't work
  - ◆ One manual connection works, but second one fails
  - ◆ Manual connections work, but automatic keying doesn't
  - ◆ IPsec works, but connections using compression fail
  - ◆ Small packets work, but large transfers fail
  - ◆ Subnet-to-subnet works, but tests from the gateways don't
- Compilation problems
  - ◆ gmp.h: No such file or directory
  - ◆ ... virtual memory exhausted
- Interpreting error messages
  - ◆ route-client (or host) exited with status 7
  - ◆ SIOCADDRT:Network is unreachable
  - ◆ ipsec\_setup: modprobe: Can't locate moduleipsec
  - ◆ ipsec\_setup: Fatal error, kernel appears to lack KLIPS
  - ◆ ipsec\_setup: ... failure to fetch key for ... from DNS
  - ◆ ipsec\_setup: ... interfaces ... and ... share address ...
  - ◆ ipsec\_setup: Cannot adjust kernel flags
  - ◆ Message numbers (MI3, QR1, et cetera) in Pluto messages
  - ◆ Connection names in Pluto error messages
  - ◆ Pluto: ... can't orient connection
  - ◆ ... we have no ipsecN interface for either end of this connection
  - ◆ Pluto: ... no connection is known
  - ◆ Pluto: ... no suitable connection ...
  - ◆ Pluto: ... no connection has been authorized
  - ◆ Pluto: ... OAKLEY DES CBC is not supported.

- ◆ Pluto: ... no acceptable transform
  - ◆ rsasigkey dumps core
  - ◆ !Pluto failure!: ... exited with ... signal 4
  - ◆ ECONNREFUSED error message
  - ◆ klips debug: ... no eroute!
  - ◆ ... trouble writing to /dev/ipsec ... SA already in use
  - ◆ ... ignoring ... payload
  - ◆ unknown parameter name "rightcert"
  - Why don't you restrict the mailing lists to reduce spam?
- 

## What is FreeS/WAN?

FreeS/WAN is a Linux implementation of the IPsec protocols, providing security services at the IP (Internet Protocol) level of the network.

For more detail, see our introduction document or the FreeS/WAN project web site.

To start setting it up, go to our quickstart guide.

Our web links document has information on IPsec for other systems.

## How do I report a problem or seek help?

*Read our troubleshooting document.*

It may guide you to a solution. If not, see its problem reporting section.

Basically, what it says is ***give us the output from ipsec barf from both gateways***. Without full information, we cannot diagnose a problem. However, *ipsec barf* produces a lot of output. If at all possible, ***please make barfs accessible via the web or FTP*** rather than sending enormous mail messages.

*Use the users mailing list for problem reports, rather than mailing developers directly.*

- ◇ This gives you access to more expertise, including users who may have encountered and solved the same problems.
- ◇ It is more likely to get a quick response. Developers may get behind on email, or even ignore it entirely for a while, but a list message (given a reasonable Subject: line) is certain to be read by a fair number of people within hours.
- ◇ It may also be important because of cryptography export laws. A US citizen who provides technical assistance to foreign cryptographic work might be charged under the arms export regulations. Such a charge would be easier to defend if the discussion took place on a public mailing list than if it were done in private mail.

*Try irc.freenode.net#freeswan.*

FreeS/WAN developers, volunteers and users can often be found there. Be patient and be prepared to provide lots of information to support your question.

If your question was really interesting, and you found an answer, please share that with the class by posting to the users mailing list. That way others with the same problem can find your answer in the archives.

*Premium support is also available.*

See the next several questions.

## Can I get ...

### Can I get an off-the-shelf system that includes FreeS/WAN?

There are a number of Linux distributions or firewall products which include FreeS/WAN. See this [list](#). Using one of these, chosen to match your requirements and budget, may save you considerable time and effort.

If you don't know your requirements, start by reading Schneier's [Secrets and Lies](#). That gives the best overview of security issues I have seen. Then consider hiring a consultant (see next question) to help define your requirements.

### Can I hire consultants or staff who know FreeS/WAN?

If you want the help of a contractor, or to hire staff with FreeS/WAN expertise, you could:

- check availability in your area through your local Linux User Group ([LUG Index](#))
- try asking on our [mailing list](#)

For companies offering support, see the next question.

### Can I get commercial support?

Many of the distributions or firewall products which include FreeS/WAN (see this [list](#)) come with commercial support or have it available as an option.

Various companies specialize in commercial support of open source software. Our project leader was a founder of the first such company, Cygnus Support. It has since been bought by [Redhat](#). Another such firm is [Linuxcare](#).

## Release questions

### What is the current release?

The current release is the highest-numbered tarball on our [distribution site](#). Almost always, any of [the mirrors](#) will have the same file, though perhaps not for a day or so after a release.

Unfortunately, the web site is not always updated as quickly as it should be.

### When is the next release?

We try to do a release approximately every six to eight weeks.

If pre-release tests fail and the fix appears complex, or more generally if the code does not appear stable when a release is scheduled, we will just skip that release.

For serious bugs, we may bring out an extra bug-fix release. These get numbers in the normal release series. For example, there was a bug found in FreeS/WAN 1.6, so we did another release less than two weeks later. The bug-fix release was called 1.7.

## Are there known bugs in the current release?

Any problems we are aware of at the time of a release are documented in the [BUGS](#) file for that release. You should also look at the [CHANGES](#) file.

Bugs discovered after a release are discussed on the [mailing lists](#). The easiest way to check for any problems in the current code would be to peruse the [List In Brief](#).

## Modifications and contributions

### Can I modify FreeS/WAN to ...?

You are free to modify FreeS/WAN in any way. See the discussion of [licensing](#) in our introduction document.

Before investing much energy in any such project, we suggest that you

- check the list of [existing patches](#)
- post something about your project to the [design mailing list](#)

This may prevent duplicated effort, or lead to interesting collaborations.

### Can I contribute to the project?

In general, we welcome contributions from the community. Various contributed patches, either to fix bugs or to add features, have been incorporated into our distribution. Other patches, not yet included in the distribution, are listed in our [web links](#) section.

Users have also contributed heavily to documentation, both by creating their own [HowTos](#) and by posting things on the [mailing lists](#) which I have quoted in these HTML docs.

There are, however, some caveats.

FreeS/WAN is being implemented in Canada, by Canadians, largely to ensure that it is entirely free of export restrictions. See this [discussion](#). We ***cannot accept code contributions from US residents or citizens***, not even one-line bugs fixes. The reasons for this were recently discussed extensively on the mailing list, in a thread starting [here](#).

Not all contributions are of interest to us. The project has a set of fairly ambitious and quite specific goals, described in our [introduction](#). Contributions that lead toward these goals are likely to be welcomed enthusiastically. Other contributions may be seen as lower priority, or even as a distraction.

Discussion of possible contributions takes place on the [design mailing list](#).

### Is there detailed design documentation?

There are:

- [RFCs](#) specifying the protocols we implement
- [man pages](#) for our utilities, library functions and file formats

- comments in the source code
- [HTML documentation](#) written primarily for users
- archived discussions from the [mailing lists](#)
- other papers mentioned in our [introduction](#)

The only formal design documents are a few papers in the last category above. All the other categories, however, have things to say about design as well.

## Will FreeS/WAN work in my environment?

### Can FreeS/WAN talk to ...?

The IPsec protocols are designed to support interoperation. In theory, any two IPsec implementations should be able to talk to each other. In practice, it is considerably more complex. We have a whole [interoperation document](#) devoted to this problem.

An important part of that document is links to the many [user-written HowTos](#) on interoperation between FreeS/WAN and various other implementations. Often the users know more than the developers about these issues (and almost always more than me :-), so these documents may be your best resource.

### Can different FreeS/WAN versions talk to each other?

Linux FreeS/WAN can interoperate with many IPsec implementations, including earlier versions of Linux FreeS/WAN itself.

In a few cases, there are some complications. See our [interoperation](#) document for details.

### Is there a limit on throughput?

There is no hard limit, but see below.

### Is there a limit on number of tunnels?

There is no hard limit, but see next question.

### Is a ... fast enough to handle FreeS/WAN with my loads?

A quick summary:

*Even a limited machine can be useful*

A 486 can handle a T1, ADSL or cable link, though the machine may be breathing hard.

*A mid-range PC (say 800 MHz with good network cards) can do a lot of IPsec*

With up to roughly 50 tunnels and aggregate bandwidth of 20 Megabits per second, it will have cycles left over for other tasks.

*There are limits*

Even a high end CPU will not come close to handling a fully loaded 100 Mbit/second Ethernet link.

Beyond about 50 tunnels it needs careful management.

See our [FreeS/WAN performance](#) document for details.

## Will FreeS/WAN work on ... ?

### Will FreeS/WAN run on my version of Linux?

We build and test on Redhat distributions, but FreeS/WAN runs just fine on several other distributions, sometimes with minor fiddles to adapt to the local environment. Details are in our [compatibility](#) document. Also, some distributions or products come with [FreeS/WAN included](#).

### Will FreeS/WAN run on non-Intel CPUs?

FreeS/WAN is *intended to run on all CPUs Linux supports*. We know of it being used in production on x86, ARM, Alpha and MIPS. It has also had successful tests on PPC and SPARC, though we don't know of actual use there. Details are in our [compatibility](#) document.

### Will FreeS/WAN run on multiprocessors?

FreeS/WAN is designed to work on any SMP architecture Linux supports, and has been tested successfully on at least dual processor Intel architecture machines. Details are in our [compatibility](#) document.

### Will FreeS/WAN work on an older kernel?

It might, but we strongly recommend using a recent 2.2 or 2.4 series kernel. Sometimes the newer versions include security fixes which can be quite important on a gateway.

Also, we use recent kernels for development and testing, so those are better tested and, if you do encounter a problem, more easily supported. If something breaks applying recent FreeS/WAN patches to an older kernel, then "update your kernel" is almost certain to be the first thing we suggest. It may be the only suggestion we have.

The precise kernel versions supported by a particular FreeS/WAN release are given in the [README](#) file of that release.

See the following question for more on kernels.

### Will FreeS/WAN run on the latest kernel version?

Sometimes yes, but quite often, no.

Kernel versions supported are given in the [README](#) file of each FreeS/WAN release. Typically, they are whatever production kernels were current at the time of our release (or shortly before; we might release for kernel  $n$  just as Linus releases  $n+1$ ). Often FreeS/WAN will work on slightly later kernels as well, but of course this cannot be guaranteed.

For example, FreeS/WAN 1.91 was released for kernels 2.2.19 or 2.4.5, the current kernels at the time. It also worked on 2.4.6, 2.4.7 and 2.4.8, but 2.4.9 had changes that caused compilation errors if it was patched with FreeS/WAN 1.91.

## Introduction to FreeS/WAN

When such changes appear, we put a fix in the FreeS/WAN snapshots, and distribute it with our next release. However, this is not a high priority for us, and it may take anything from a few days to several weeks for such a problem to find its way to the top of our kernel programmer's To-Do list. In the meanwhile, you have two choices:

- either stick with a slightly older kernel, even if it is not the latest and greatest. This is recommended for production systems; new versions may have new bugs.
- or fix the problem yourself and send us a patch, via the [Users mailing list](#).

We don't even try to keep up with kernel changes outside the main 2.2 and 2.4 branches, such as the 2.4.x-ac patched versions from Alan Cox or the 2.5 series of development kernels. We'd rather work on developing the FreeS/WAN code than on chasing these moving targets. We are, however, happy to get patches for problems discovered there.

See also the [Choosing a kernel](#) section of our installation document.

## Will FreeS/WAN work on unusual network hardware?

IPsec is designed to work over any network that IP works over, and FreeS/WAN is intended to work over any network interface hardware that Linux supports.

If you have working IP on some unusual interface — perhaps Arcnet, Token Ring, ATM or Gigabit Ethernet — then IPsec should "just work".

That said, practice is sometimes less tractable than theory. Our testing is done almost entirely on:

- 10 or 100 Mbit Ethernet
- ADSL or cable connections, with and without PPPoE
- IEEE 802.11 wireless LANs (see [below](#))

If you have some other interface, especially an uncommon one, it is entirely possible you will get bitten either by a FreeS/WAN bug which our testing did not turn up, or by a bug in the driver that shows up only with our loads.

If IP works on your interface and FreeS/WAN doesn't, seek help on the [mailing lists](#).

Another FAQ section describes [MTU problems](#). These are a possibility for some interfaces.

## Will FreeS/WAN work on a VLAN (802.1q) network?

Yes, FreeS/WAN works fine, though some network drivers have problems with jumbo sized ethernet frames. If you used `interfaces=%defaultroute` you do not need to change anything, but if you specified an interface (eg `eth0`) then remember you must change that to reflect the VLAN interface (eg `eth0.2` for VLAN ID 2).

The "eepro100" module is known to be broken, use the `e100` driver for those cards instead (included in 2.4 as 'alternative driver' for the Intel EtherExpressPro/100).

You do not need to change any MTU setting (those are workarounds that are only needed for buggy drivers)

*This FAQ contributed by Paul Wouters.*

## Does FreeS/WAN support ...

For a discussion of which parts of the IPsec specifications FreeS/WAN does and does not implement, see our [compatibility](#) document.

For information on some often-requested features, see below.

### Does FreeS/WAN support site-to-site VPN (Virtual Private Network) applications?

Absolutely. See this FreeS/WAN-FreeS/WAN [configuration example](#). If only one site is using FreeS/WAN, there may be a relevant HOWTO on our [interop](#) page.

### Does FreeS/WAN support remote users connecting to a LAN?

Yes. We call the remote users "Road Warriors". Check out our FreeS/WAN-FreeS/WAN [Road Warrior Configuration Example](#).

If your Road Warrior is a Windows or Mac PC, you may need to install an IPsec implementation on that machine. Our [interop](#) page lists many available brands, and features links to several HOWTOs.

### Does FreeS/WAN support remote users using shared secret authentication?

*Yes, but there are severe restrictions, so **we strongly recommend using RSA keys for authentication instead.***

See this [FAQ question](#).

### Does FreeS/WAN support wireless networks?

Yes, it is a common practice to use IPsec over wireless networks because their built-in encryption, WEP, is insecure.

There is some [discussion](#) in our advanced configuration document. See also the [WaveSEC site](#).

### Does FreeS/WAN support X.509 or other PKI certificates?

Vanilla FreeS/WAN does not support X.509, but Andreas Steffen and others have provided a popular, well-supported X.509 patch.

- [patch](#)
- [Super FreeS/WAN](#) incorporates this and other user-contributed patches.
- Kai Martius' [X.509 Installation and Configuration Guide](#)

Linux FreeS/WAN features [Opportunistic Encryption](#), an alternative Public Key Infrastructure based on Secure DNS.

## Does FreeS/WAN support user authentication (Radius, SecureID, Smart Card...)?

Andreas Steffen's [X.509 patch](#) (v. 1.42+) supports Smart Cards. The patch does not ship with vanilla FreeS/WAN, but will be incorporated into [Super FreeS/WAN 2.01+](#). The patch implements the PKCS#15 Cryptographic Token Information Format Standard, using the OpenSC smartcard library functions.

Older news:

A user-supported patch to FreeS/WAN 1.3, for smart card style authentication, is available on [Bastiaan's site](#). It supports skeyid and ibutton. This patch is not part of Super FreeS/WAN.

For a while progress on this front was impeded by a lack of standard. The IETF [working group](#) has now nearly completed its recommended solution to the problem; meanwhile several vendors have implemented various things.

Of course, there are various ways to avoid any requirement for user authentication in IPsec. Consider the situation where road warriors build IPsec tunnels to your office net and you are considering requiring user authentication during tunnel negotiation. Alternatives include:

- If you can trust the road warrior machines, then set them up so that only authorised users can create tunnels. If your road warriors use laptops, consider the possibility of theft.
- If the tunnel only provides access to particular servers and you can trust those servers, then set the servers up to require user authentication.

If either of those is trustworthy, it is not clear that you need user authentication in IPsec.

## Does FreeS/WAN support NAT traversal?

Vanilla FreeS/WAN does not, but thanks to Mathieu Lafon and Arkoon Network Security, there's a patch to support this.

- [patch and documentation](#)
- [Super FreeS/WAN](#) incorporates this and other user-contributed patches.

The NAT traversal patch has some issues with PSKs, so you may wish to authenticate with RSA keys, or X.509 (requires a patch which is also included in Super FreeS/WAN). Doing the latter also has advantages when dealing with large numbers of clients who may be behind NAT; instead of having to make an individual Roadwarrior connection for each virtual IP, you can use the "rightsubnetwithin" parameter to specify a range. See [these rightsubnetwithin instructions](#).

## Does FreeS/WAN support assigning a "virtual identity" to a remote system?

Some IPsec implementations allow you to make the source address on packets sent by a Road Warrior machine be something other than the address of its interface to the Internet. This is sometimes described as assigning a virtual identity to that machine.

FreeS/WAN does not directly support this, but it can be done. See this [FAQ question](#).

## Does FreeS/WAN support single DES encryption?

*No*, single DES is not used either at the IKE level for negotiating connections or at the IPsec level for actually building them.

Single DES is insecure. As we see it, it is more important to deliver real security than to comply with a standard which has been subverted into allowing use of inadequate methods. See this discussion.

If you want to interoperate with an IPsec implementation which offers only DES, see our interoperation document.

## Does FreeS/WAN support AES encryption?

AES is a new US government block cipher standard to replace the obsolete DES.

At time of writing (March 2002), the FreeS/WAN distribution does not yet support AES but user-written patches are available to add it. Our kernel programmer is working on integrating those patches into the distribution, and there is active discussion of this on the design mailing list.

## Does FreeS/WAN support other encryption algorithms?

Currently triple DES is the only cipher supported. AES will almost certainly be added (see previous question), and it is likely that in the process we will also add the other two AES finalists with open licensing, Twofish and Serpent.

We are extremely reluctant to add other ciphers. This would make both use and maintenance of FreeS/WAN more complex without providing any clear benefit. Complexity is emphatically not desirable in a security product.

Various users have written patches to add other ciphers. We provide links to these.

## Can I ...

### Can I use policy groups along with explicitly configured connections?

Yes, you can, so long as you pay attention to the selection rule, which can be summarized "the most specific connection wins". We describe the rule in our policy groups document, and provide a more technical explanation in man ipsec.conf.

A good guideline: If you have a regular connection defined in *ipsec.conf*, ensure that a subset of that connection is not listed in a less restrictive policy group. Otherwise, FreeS/WAN will use the subset, with its more specific source/destination pair.

Here's an example. Suppose you are the system administrator at 192.0.2.2. You have this connection in *ipsec.conf*: *ipsec.conf*:

```
conn net-to-net
    left=192.0.2.2           # you are here
    right=192.0.2.8
    rightsubnet=192.0.2.96/27
```

....

If you then place a host or net within *rightsubnet*, (let's say 192.0.2.98) in *private-or-clear*, you may find that 192.0.2.2 at times communicates in the clear with 192.0.2.98. That's consistent with the rule, but may be contrary to your expectations.

On the other hand, it's safe to put a larger subnet in a less restrictive policy group file. If *private-or-clear* contains 192.0.2.0/24, then the more specific *net-to-net* connection is used for any communication to 192.0.2.96/27. The more general policy applies only to communication with hosts or subnets in 192.0.2.0/24 without a more specific policy or connection.

## Can I turn off policy groups?

Yes. Use [these instructions](#).

## Can I reload connection info without restarting?

Yes, you can do this. Here are the details, in a mailing list message from Pluto programmer Hugh Redelmeier:

```
| How can I reload config's without restarting all of pluto and klips?  I am using
| FreeSWAN -> PGPNet in a medium sized production environment, and would like to be
| able to add new connections ( i am using include config/* ) without dropping current
| SA's.
|
| Can this be done?
|
| If not, are there plans to add this kind of feature?
```

```
        ipsec auto --add whatever
This will look in the usual place (/etc/ipsec.conf) for a conn named
whatever and add it.
```

```
If you added new secrets, you need to do
        ipsec auto --rereadsecrets
before Pluto needs to know those secrets.
```

```
| I have looked (perhaps not thoroughly enough tho) to see how to do this:
```

```
There may be more bits to look for, depending on what you are trying
to do.
```

Another useful command here is *ipsec auto --replace <conn\_name>* which re-reads data for a named connection.

## Can I use several masqueraded subnets?

Yes. This is done all the time. See the discussion in our [setup](#) document. The only restriction is that the subnets on the two ends must not overlap. See the next question.

Here is a mailing list message on the topic. The user incorrectly thinks you need a 2.4 kernel for this — actually various people have been doing it on 2.0 and 2.2 for quite some time — but he has it right for 2.4.

```
Subject: Double NAT and freeswan working :)
Date: Sun, 11 Mar 2001
From: Paul Wouters <paul@xtdnet.nl>
```

## Introduction to FreeS/WAN

Just to share my pleasure, and make an entry for people who are searching the net on how to do this. Here's the very simple solution to have a double NAT'ed network working with freeswan. (Not sure if this is old news, but I'm not on the list (too much spam) and I didn't read this in any HOWTO/FAQ/doc on the freeswan site yet (Sandy, put it in! :))

```
10.0.0.0/24 --- 10.0.0.1 a.b.c.d ---- a.b.c.e {internet} ----+
                                                                |
10.0.1.0/24 --- 10.0.1.1 f.g.h.i ---- f.g.h.j {internet} ----+
```

the goal is to have the first network do a VPN to the second one, yet also have NAT in place for connections not destined for the other side of the NAT. Here the two Linux security gateways have one real IP number (cable modem, dialup, whatever).

The problem with NAT is you don't want packets from 10.\*.\*.\* to 10.\*.\*.\* to be NAT'ed. While with Linux 2.2, you can't, with Linux 2.4 you can.

(This has been tested and works for 2.4.2 with Freeswan snapshot2001mar8b)

relevant parts of /etc/ipsec.conf:

```
left=f.g.h.i
leftsubnet=10.0.1.0/24
leftnexthop=f.g.h.j
leftfirewall=yes
leftid=@firewall.netone.nl
lefttrsasigkey=0x0.....
right=a.b.c.d
rightsubnet=10.0.0.0/24
rightnexthop=a.b.c.e
rightfirewall=yes
rightid=@firewall.nettwo.nl
righttrsasigkey=0x0.....
# To authorize this connection, but not actually start it, at startup,
# uncomment this.
auto=add
```

and now the real trick. Setup the NAT correctly on both sites:

```
iptables -t nat -F
iptables -t nat -A POSTROUTING -o eth0 -d \! 10.0.0.0/8 -j MASQUERADE
```

This tells the NAT code to only do NAT for packets with destination other than 10.\* networks. Note the backslash to mask the exclamation mark to protect it against the shell.

Happy painting :)

Paul

## Can I use subnets masqueraded to the same addresses?

**No.** The notion that IP addresses are unique is one of the fundamental principles of the IP protocol. Messing with it is exceedingly perilous.

Fairly often a situation comes up where a company has several branches, all using the same non-routable addresses, perhaps 192.168.0.0/24. This works fine as long as those nets are kept distinct. The IP masquerading on their firewalls ensures that packets reaching the Internet carry the firewall address, not the

private address.

This can break down when IPsec enters the picture. FreeS/WAN builds a tunnel that pokes through both masquerades and delivers packets from *leftsubnet* to *rightsubnet* and vice versa. For this to work, the two subnets *must* be distinct.

There are several solutions to this problem.

Usually, you **re-number the subnets**. Perhaps the Vancouver office becomes 192.168.101.0/24, Calgary 192.168.102.0/24 and so on. FreeS/WAN can happily handle this. With, for example *leftsubnet=192.168.101.0/24* and *rightsubnet=192.168.102.0/24* in a connection description, any machine in Calgary can talk to any machine in Vancouver. If you want to be more restrictive and use something like *leftsubnet=192.168.101.128/25* and *rightsubnet=192.168.102.240/28* so only certain machines on each end have access to the tunnel, that's fine too.

You could also **split the subnet** into smaller ones, for example using *192.168.1.0/25* in Vancouver and *rightsubnet=192.168.0.128/25* in Calgary.

Alternately, you can just **give up routing** directly to machines on the subnets. Omit the *leftsubnet* and *rightsubnet* parameters from your connection descriptions. Your IPsec tunnels will then run between the public interfaces of the two firewalls. Packets will be masqueraded both before they are put into tunnels and after they emerge. Your Vancouver client machines will see only one Calgary machine, the firewall.

### Can I assign a road warrior an address on my net (a virtual identity)?

Often it would be convenient to be able to give a Road Warrior an IP address which appears to be on the local network. Some IPsec implementations have support for this, sometimes calling the feature "virtual identity".

Currently (Sept 2002) FreeS/WAN does not support this, and we have no definite plans to add it. The difficulty is that is not yet a standard mechanism for it. There is an Internet Draft for a method of doing it using DHCP which looks promising. FreeS/WAN may support that in a future release.

In the meanwhile, you can do it yourself using the Linux iproute2(8) facilities. Details are in this paper.

Another method has also been discussed on the mailing list.:

- You can use a variant of the extruded subnet procedure.
- You have to avoid having the road warrior's assigned address within the range you actually use at home base. See previous question.
- On the other hand, you want the roadwarrior's address to be within the range that *seems* to be on your network.

For example, you might have:

*leftsubnet=a.b.c.0/25*

head office network

*rightsubnet=a.b.c.129/32*

extruded to a road warrior. Note that this is not in a.b.c.0/25

*a.b.c.0/24*

whole network, including both the above

You then set up routing so that the office machines use the IPsec gateway as their route to a.b.c.128/25. The leftsubnet parameter tells the road warriors to use tunnels to reach a.b.c.0/25, so you should have two-way communication. Depending on your network and applications, there may be some additional work to do on DNS or Windows configuration

### Can I support many road warriors with one gateway?

Yes. This is easily done, using

*either RSA authentication*  
standard in the FreeS/WAN distribution  
*or X.509 certificates*  
requires Super FreeS/WAN or a patch.

In either case, each Road Warrior must have a different key or certificate.

It is also possible using pre-shared key authentication, though we don't recommend this; see the next question for details.

If you expect to have more than a few dozen Road Warriors connecting simultaneously, you may need a fairly powerful gateway machine. See our document on FreeS/WAN performance.

### Can I have many road warriors using shared secret authentication?

*Yes, but avoid it if possible.*

You can have multiple Road Warriors using shared secret authentication *only if they all use the same secret*. You must also set:

```
uniqueids=no
```

in the connection definition.

Why it's less secure:

- If you have many users, it becomes almost certain the secret will leak
- The secret becomes quite valuable to an attacker
- All users authenticate the same way, so the gateway cannot tell them apart for logging or access control purposes
- Changing the secret is difficult. You have to securely notify all users.
- If you find out the secret has been compromised, you can change it, but then what? None of your users can connect without the new secret. How will you notify them all, quickly and securely, without using the VPN?

This is a designed-in limitation of the IKE key negotiation protocol, not a problem with our implementation.

***We very strongly recommend that you avoid using shared secret authentication for multiple Road Warriors. Use RSA authentication instead.***

The longer story: When using shared secrets, the protocol requires that the responding gateway be able to

determine which secret to use at a time when all it knows about the initiator is an IP address. This works fine if you know the initiator's address in advance and can use it to look up the appropriate secret. However, it fails for Road Warriors since the gateway cannot know their IP addresses in advance.

With RSA signatures (or certificates) the protocol is slightly different. The initiator provides an identifier early in the exchange and the responder can use that identifier to look up the correct key or certificate. See [above](#).

## Can I use Quality of Service routing with FreeS/WAN?

From project technical lead Henry Spencer:

```
> Do QoS add to FreeS/WAN?  
> For example integrating DiffServ and FreeS/WAN?
```

With a current version of FreeS/WAN, you will have to add `hidetos=no` to the `config-setup` section of your configuration file. By default, the TOS field of tunnel packets is zeroed; with `hidetos=no`, it is copied from the packet inside. (This is a modest security hole, which is why it is no longer the default.)

DiffServ does not interact well with tunneling in general. Ways of improving this are being studied.

Copying the TOS (type of service) information from the encapsulated packet to the outer header reveals the TOS information to an eavesdropper. This does not tell him much, but it might be of use in [traffic analysis](#). Since we do not have to give it to him, our default is not to.

Even with the TOS hidden, you can still:

- apply QOS rules to the tunneled (ESP) packets; for example, by giving ESP packets a certain priority.
- apply QOS rules to the packets as they enter or exit the tunnel via an IPsec virtual interface (eg. `ipsec0`).

See [ipsec.conf\(5\)](#) for more on the `hidetos=` parameter.

## Can I recognise dead tunnels and shut them down?

There is no general mechanism to do this in the IPsec protocols.

From time to time, there is discussion on the IETF Working Group [mailing list](#) of adding a "keep-alive" mechanism (which some say should be called "make-dead"), but it is a fairly complex problem and no consensus has been reached on whether or how it should be done.

The protocol does have optional delete-SA messages which one side can send when it closes a connection in hopes this will cause the other side to do the same. FreeS/WAN does not currently support these. In any case, they would not solve the problem since:

- a gateway that crashes or hangs would not send the messages
- the sender is not required to send them
- they are not authenticated, so any receiver that trusts them leaves itself open to a [denial of service](#) attack

## Introduction to FreeS/WAN

- the receiver is not required to do anything about them
- the receiver cannot acknowledge them; the protocol provides no mechanism for that
- since they are not acknowledged, the sender cannot rely on them

However, connections do have limited lifetimes and you can control how many attempts your gateway makes to rekey before giving up. For example, you can set:

```
conn default
    keyingtries=3
    keylife=30m
```

With these settings old connections will be cleaned up. Within 30 minutes of the other end dying, rekeying will be attempted. If it succeeds, the new connection replaces the old one. If it fails, no new connection is created. Either way, the old connection is taken down when its lifetime expires.

Here is a mailing list message on the topic from FreeS/WAN tech support person Claudia Schmeing:

You ask how to determine whether a tunnel is redundant:

```
> Can anybody explain the best way to determine this. Esp when a RW has
> disconnected? I thought 'ipsec auto --status' might be one way.
```

If a tunnel goes down from one end, Linux FreeS/WAN on the other end has no way of knowing this until it attempts to rekey. Once it tries to rekey and fails, it will 'know' that the tunnel is down.

Because it doesn't have a way of knowing the state until this point, it will also not be able to tell you the state via `ipsec auto --status`.

```
> However, comparing output from a working tunnel with that of one that
> was closed
> did not show clearly show tunnel status.
```

If your tunnel is down but not 'unrouted' (see `man ipsec_auto`), you should not be able to ping the opposite side of the tunnel. You can use this as an indicator of tunnel status.

On a related note, you may be interested to know that as of 1.7, redundant tunnels caused by RW disconnections are likely to be less of a pain. From `doc/CHANGES`:

```
There is a new configuration parameter, uniqueids, to control a new Pluto
option: when a new connection is negotiated with the same ID as an old
one, the old one is deleted immediately. This should help eliminate
dangling Road Warrior connections when the same Road Warrior reconnects.
It thus requires that IDs not be shared by hosts (a previously legal but
probably useless capability). NOTE WELL: the sample ipsec.conf now has
uniqueids=yes in its config-setup section.
```

Cheers,

Claudia

## Can I build IPsec tunnels over a demand-dialed link?

This is possible, but not easy. FreeS/WAN technical lead Henry Spencer wrote:

```
> 5. If the ISDN link goes down in between and is reestablished, the SAs
> are still up but the routes are deleted and the IPsec interface shows
> garbage (with ifconfig)
> 6. Only restarting IPsec will bring the VPN back online.
```

This one is awkward to solve. If the real interface that the IPsec interface is mounted on goes down, it takes most of the IPsec machinery down with it, and a restart is the only good way to recover.

The only really clean fix, right now, is to split the machines in two:

1. A minimal machine serves as the network router, and only it is aware that the link goes up and down.
2. The IPsec is done on a separate gateway machine, which thinks it has a permanent network connection, via the router.

This is clumsy but it does work. Trying to do both functions within a single machine is tricky. There is a software package (diald) which will give the illusion of a permanent connection for demand-dialed modem connections; I don't know whether it's usable for ISDN, or whether it can be made to cooperate properly with FreeS/WAN.

Doing a restart each time the interface comes up *\*does\** work, although it is a bit painful. I did that with PPP when I was running on a modem link; it wasn't hard to arrange the PPP scripts to bring IPsec up and down at the right times. (I'd meant to investigate diald but never found time.)

In principle you don't need to do a complete restart on reconnect, but you do have to rebuild some things, and we have no nice clean way of doing only the necessary parts.

In the same thread, one user commented:

```
Subject: Re: linux-ipsec: IPsec and Dial Up Connections
Date: Wed, 22 Nov 2000
From: Andy Bradford <andyb@calderasystems.com>
```

On Wed, 22 Nov 2000 19:47:11 +0100, Philip Reetz wrote:

```
> Are there any ideas what might be the cause of the problem and any way
> to work around it.
> Any help is highly appreciated.
```

On my laptop, when using ppp there is a ip-up script in /etc/ppp that will be executed each time that the ppp interface is brought up. Likewise there is an ip-down script that is called when it is taken down. You might consider customizing those to stop and start FreeS/WAN with each connection. I believe that ISDN uses the same files, though I could be wrong---there should be something similar though.

## Can I build GRE, L2TP or PPTP tunnels over IPsec?

Yes. Normally this is not necessary, but it is useful in a few special cases. For example, if you must route non-IP packets such as IPX, you will need to use a tunneling protocol that can route these packets. IPsec can

be layered around it for extra security. Another example: you can provide failover protection for high availability (HA) environments by combining IPsec with other tools. Ken Bantoft describes one such setup in [Using FreeS/WAN with Linux-HA, GRE, OSPF and BGP for enterprise grade VPN solutions](#).

GRE over IPsec is covered as part of [that document](#). [Here are links](#) to other GRE resources. Jacco de Leuw has created [this page on L2TP over IPsec](#) with instructions for FreeS/WAN and several other brands of IPsec software.

Please let us know of other useful links via the [mailing lists](#).

### ... use Network Neighborhood (Samba, NetBIOS) over IPsec?

Your local PC needs to know how to translate NetBIOS names to IP addresses. It may do this either via a local LMHOSTS file, or using a local or remote WINS server. The WINS server is preferable since it provides a centralized source of the information to the entire network. To use a WINS server over the [VPN](#) (or any IP-based network), you must enable "NetBIOS over TCP".

[Samba](#) can emulate a WINS server on Linux.

See also several discussions in our [September 2002 Users archives](#)

## Life's little mysteries

FreeS/WAN is a fairly complex product. (Neither the networks it runs on nor the protocols it uses are simple, so it could hardly be otherwise.) It therefore sometimes exhibits behaviour which can be somewhat confusing, or has problems which are not easy to diagnose. This section tries to explain those problems.

Setup and configuration of FreeS/WAN are covered in other documentation sections:

- [basic setup and configuration](#)
- [advanced configuration](#)
- [Troubleshooting](#)

However, we also list some of the commonest problems here.

### I cannot ping ....

This question is dealt with in the advanced configuration section under the heading [multiple tunnels](#).

The standard subnet-to-subnet tunnel protects traffic *only between the subnets*. To test it, you must use pings that go from one subnet to the other.

For example, suppose you have:

```
subnet a.b.c.0/24
      |
eth1 = a.b.c.1
      |
      |
eth0 = 192.0.2.8
      |
```

## Introduction to FreeS/WAN

```
~ internet ~  
  
      |  
eth0 = 192.0.2.11  
      |  
      | gate2  
eth1 = x.y.z.1  
      |  
      | subnet x.y.z.0/24
```

and the connection description:

```
conn abc-xyz  
left=192.0.2.8  
leftsubnet=a.b.c.0/24  
right=192.0.2.11  
rightsubnet=x.y.z.0/24
```

You can test this connection description only by sending a ping that will actually go through the tunnel. Assuming you have machines at addresses a.b.c.2 and x.y.z.2, pings you might consider trying are:

*ping from x.y.z.2 to a.b.c.2 or vice versa*

Succeeds if tunnel is working. This is the *only valid test of the tunnel*.

*ping from gate2 to a.b.c.2 or vice versa*

**Does not use tunnel.** gate2 is not on protected subnet.

*ping from gate1 to x.y.z.2 or vice versa*

**Does not use tunnel.** gate1 is not on protected subnet.

*ping from gate1 to gate2 or vice versa*

**Does not use tunnel.** Neither gate is on a protected subnet.

Only the first of these is a useful test of this tunnel. The others do not use the tunnel. Depending on other details of your setup and routing, they:

- either fail, telling you nothing about the tunnel
- or succeed, telling you nothing about the tunnel since these packets use some other route

In some cases, you may be able to get around this. For the example network above, you could use:

```
ping -I a.b.c.1 x.y.z.1
```

Both the addresses given are within protected subnets, so this should go through the tunnel.

If required, you can build additional tunnels so that all the machines involved can talk to all the others. See [multiple tunnels](#) in the advanced configuration document for details.

## It takes forever to ...

Users fairly often report various problems involving long delays, sometimes on tunnel setup and sometimes on operations done through the tunnel, occasionally on simple things like ping or more often on more complex operations like doing NFS or Samba through the tunnel.

Almost always, these turn out to involve failure of a DNS lookup. The timeouts waiting for DNS are typically set long so that you won't time out when a query involves multiple lookups or long paths. Genuine failures therefore produce long delays before they are detected.

It takes forever to ...

## Introduction to FreeS/WAN

A mailing list message from project technical lead Henry Spencer:

```
> ... when i run /etc/rc.d/init.d/ipsec start, i get:  
> ipsec_setup: Starting FreeS/WAN IPsec 1.5...  
> and it just sits there, doesn't give back my bash prompt.
```

Almost certainly, the problem is that you're using DNS names in your `ipsec.conf`, but DNS lookups are not working for some reason. You will get your prompt back... eventually. But the DNS timeouts are long. Doing something about this is on our list, but it is not easy.

In the meanwhile, we recommend that connection descriptions in [`ipsec.conf\(5\)`](#) use numeric IP addresses rather than names which will require a DNS lookup.

Names that do not require a lookup are fine. For example:

- a road warrior might use the identity `rightid=@lancelot.example.org`
- the gateway might use `leftid=@camelot.example.org`

These are fine. The @ sign prevents any DNS lookup. However, do not attempt to give the gateway address as `left=camelot.example.org`. That requires a lookup.

A post from one user after solving a problem with long delays:

```
Subject: Final Answer to Delay!!!  
Date: Mon, 19 Feb 2001  
From: "Felippe Solutions" <felippe@solutionstecnologia.com.br>
```

Sorry people, but seems like the Delay problem had nothing to do with freeswan.

The problem was DNS as some people sad from the beginning, but not the way they thought it was happening. Samba, ssh, telnet and other apps try to reverse lookup addresses when you use IP numbers (Stupid that ahh).

I could ping very fast because I always ping with "-n" option, but I don't know the option on the other apps to stop reverse addressing so I don't use it.

This post is fairly typical. These problems are often tricky and frustrating to diagnose, and most turn out to be DNS-related.

One suggestion for diagnosis: test with both names and addresses if possible. For example, try all of:

- ping *address*
- ping -n *address*
- ping *name*

If these behave differently, the problem must be DNS-related since the three commands do exactly the same thing except for DNS lookups.

## I send packets to the tunnel with route(8) but they vanish

IPsec connections are designed to carry only packets travelling between pre-defined connection endpoints. As

project technical lead Henry Spencer put it:

IPsec tunnels are not just virtual wires; they are virtual wires with built-in access controls. Negotiation of an IPsec tunnel includes negotiation of access rights for it, which don't include packets to/from other IP addresses. (The protocols themselves are quite inflexible about this, so there are limits to what we can do about it.)

For fairly obvious security reasons, and to comply with the IPsec RFCs, **KLIPS** drops any packets it receives that are not allowed on the tunnels currently defined. So if you send it packets with *route(8)*, and suitable tunnels are not defined, the packets vanish. Whether this is reported in the logs depends on the setting of *klipsdebug* in your *ipsec.conf(5)* file.

To rescue vanishing packets, you must ensure that suitable tunnels for them exist, by editing the connection descriptions in *ipsec.conf(5)*. For example, supposing you have a simple setup:

```
leftsubnet -- leftgateway === internet === roadwarrior
```

If you want to give the roadwarrior access to some resource that is located behind the left gateway but is not in the currently defined left subnet, then the usual procedure is to define an additional tunnel for those packets by creating a new connection description.

In some cases, it may be easier to alter an existing connection description, enlarging the definition of *leftsubnet*. For example, instead of two connection descriptions with 192.168.8.0/24 and 192.168.9.0/24 as their *leftsubnet* parameters, you can use a single description with 192.168.8.0/23.

If you have multiple endpoints on each side, you need to ensure that there is a route for each pair of endpoints. See this [example](#).

## When a tunnel goes down, packets vanish

This is a special case of the vanishing packet problem described in the previous question. Whenever **KLIPS** sees packets for which it does not have a tunnel, it drops them.

When a tunnel goes away, either because negotiations with the other gateway failed or because you gave an *ipsec auto --down* command, the route to its other end is left pointing into **KLIPS**, and **KLIPS** will drop packets it has no tunnel for.

This is a documented design decision, not a bug. FreeS/WAN must not automatically adjust things to send packets via another route. The other route might be insecure.

Of course, re-routing may be necessary in many cases. In those cases, you have to do it manually or via scripts. We provide the *ipsec auto --unroute* command for these cases.

From *ipsec auto(8)*:

Normally, pluto establishes a route to the destination specified for a connection as part of the *--up* operation. However, the route and only the route can be established with the *--route* operation. Until and unless an actual connection is established, this discards any packets sent there, which may be preferable to having them sent elsewhere based on a more general route (e.g., a default route).

## Introduction to FreeS/WAN

Normally, pluto's route to a destination remains in place when a `---down` operation is used to take the connection down (or if connection setup, or later automatic rekeying, fails). This permits establishing a new connection (perhaps using a different specification; the route is altered as necessary) without having a "window" in which packets might go elsewhere based on a more general route. Such a route can be removed using the `---unroute` operation (and is implicitly removed by `---delete`).

See also this mailing list [message](#).

## The firewall ate my packets!

If firewalls filter out:

- either the UDP port 500 packets used in IKE negotiations
- or the ESP and AH (protocols 50 and 51) packets used to implement the IPsec tunnel

then IPsec cannot work. The first thing to check if packets seem to be vanishing is the firewall rules on the two gateway machines and any other machines along the path that you have access to.

For details, see our document on [firewalls](#).

Some advice from technical lead Henry Spencer on diagnosing such problems:

```
> > Packets vanishing between the hardware interface and the ipsecN interface
> > is usually the result of firewalls not being configured to let them in...
>
> Thanks for the suggestion. If only it were that simple! My ipchains startup
> script does take care of that, but just in case I manually inserted rules
> accepting everything from london on dublin. No difference.
```

The other thing to check is whether the "RX packets dropped" count on the ipsecN interface (run `"ifconfig ipsecN"`, for N=1 or whatever, to see the counts) is rising. If so, then there's some sort of configuration mismatch between the two ends, and IPsec itself is rejecting them. If none of the ipsecN counts is rising, then the packets are never reaching the IPsec machinery, and the problem is almost certainly in firewalls etc.

## Dropped connections

Networks being what they are, IPsec connections can be broken for any number of reasons, ranging from hardware failures to various software problems such as the path MTU problems discussed [elsewhere in the FAQ](#). Fortunately, various diagnostic tools exist that help you sort out many of the possible problems.

There is one situation, however, where FreeS/WAN (using default settings) may destroy a connection for no readily apparent reason. This occurs when things are *misconfigured* so that *two tunnels* from the same gateway expect *the same subnet on the far end*.

In this situation, the first tunnel comes up fine and works until the second is established. At that point, because of the way we track connections internally, the first tunnel ceases to exist as far as this gateway is concerned. Of course the far end does not know that, and a storm of error messages appears on both systems as it tries to use the tunnel.

If the far end gives up, goes back to square one and negotiates a new tunnel, then that wipes out the second tunnel and ...

The solution is simple. ***Do not build multiple conn descriptions with the same remote subnet.***

This is actually intended to be a feature, rather than a bug. Consider the situation where a single remote system goes down, then comes back up and reconnects to the gateway. It is useful to have the gateway tear down the old tunnel and recover resources when the reconnection is made. It recognises that situation by checking the remote subnet for each tunnel it builds and discarding duplicates. This works fine as long as you don't configure multiple tunnels with the same remote subnet.

If this behaviour is inconvenient for you, you can disable it by setting *uniqueids=no* in `ipsec.conf(5)`.

## Disappearing %defaultroute

When an underlying connection (eg. ppp) goes down, FreeS/WAN will not recover properly without a little help. Here are the symptoms that FreeS/WAN user Michael Carmody noticed:

```
> After about 24 hours the freeswan connection takes over the default route.
>
> i.e instead of default gateway pointing to the router via eth0, it becomes a
> pointer to the router via ipsec0.

> All internet access is then lost as all replies (and not just the link I
> wanted) are routed out ipsec0 and the router doesn't respond to the ipsec
> traffic.
```

If you're using a FreeS/WAN 2.x/KLIPS system, simply re-attach the IPsec virtual interface with *ipsec tnconfig* command such as:

```
ipsec tnconfig --attach --virtual ipsec0 --physical ppp0
```

In your command, name the physical and virtual interfaces as they appear paired on your system during regular uptime. For a system with several physical/virtual interface pairs on flaky links, you'll need more than one such command. If you're using FreeS/WAN 1.x, you must restart FreeS/WAN, which is more time consuming.

Here is a script which can help to automate the process of FreeS/WAN restart at need. It could easily be adapted to use *tnconfig* instead.

## TCPdump on the gateway shows strange things

As another user pointed out, keeping the connect

Attempting to look at IPsec packets by running monitoring tools on the IPsec gateway machine can produce silly results. That machine is mangling the packets for IPsec, and possibly for firewall or NAT purposes as well. If the internals of the machine's IP stack are not what the monitoring tool expects, then the tool can misinterpret them and produce nonsense output.

See our testing document for more detail.

## Traceroute does not show anything between the gateways

As far as traceroute can see, the two gateways are one hop apart; the data packet goes directly from one to the other through the tunnel. Of course the outer packets that implement the tunnel pass through whatever lies between the gateways, but those packets are built and dismantled by the gateways. Traceroute does not see them and cannot report anything about their path.

Here is a mailing list message with more detail.

```
Date: Mon, 14 May 2001
To: linux-ipsec@freeswan.org
From: "John S. Denker" <jsd@research.att.com>
Subject: Re: traceroute: one virtual hop
```

```
At 02:20 PM 5/14/01 -0400, Claudia Schmeing wrote:
>
>> > A bonus question: traceroute in subnet to subnet enviroment looks like:
>> >
>> > traceroute to andris.dmz (172.20.24.10), 30 hops max, 38 byte packets
>> > 1  drama (172.20.1.1)  0.716 ms  0.942 ms  0.434 ms
>> > 2  * * *
>> > 3  andris.dmz (172.20.24.10)  73.576 ms  78.858 ms  79.434 ms
>> >
>> > Why aren't there the other hosts which take part in the delivery during
>   * * * ?
>
>If there is an ipsec tunnel between GateA and Gate B, this tunnel forms a
>'virtual wire'.  When it is tunneled, the original packet becomes an inner
>packet, and new ESP and/or AH headers are added to create an outer packet
>around it. You can see an example of how this is done for AH at
>doc/ipsec.html#AH . For ESP it is similar.
>
>Think about the packet's path from the inner packet's perspective.
>It leaves the subnet, goes into the tunnel, and re-emerges in the second
>subnet. This perspective is also the only one available to the
>'traceroute' command when the IPSec tunnel is up.
```

Claudia got this exactly right. Let me just expand on a couple of points:

\*) GateB is exactly one (virtual) hop away from GateA. This is how it would be if there were a physically private wire from A to B. The virtually private connection should work the same, and it does.

\*) While the information is in transit from GateA to GateB, the hop count of the outer header (the "envelope") is being decremented. The hop count of the inner header (the "contents" of the envelope) is not decremented and should not be decremented. The hop count of the outer header is not derived from and should not be derived from the hop count of the inner header.

Indeed, even if the packets did time out in transit along the tunnel, there would be no way for traceroute to find out what happened. Just as information cannot leak out of the tunnel to the outside, information cannot leak into the tunnel from outside, and this includes ICMP messages from routers along the path.

There are some cases where one might wish for information about what is happening at the IP layer (below the tunnel layer) -- but the protocol makes no provision for this. This raises all sorts of conceptual issues. AFAIK nobody has ever cared enough to really figure out what should happen, let alone implement it and standardize it.

\*) I consider the "\*" to be a slight bug. One might wish for it to be replaced by "GateB GateB GateB". It has to do with treating host-to-subnet traffic different from subnet-to-subnet traffic (and other gory details). I fervently hope KLIPS2 will make this problem go away.

\*) If you want to ask questions about the link from GateA to GateB at the IP level (below the tunnel level), you have to ssh to GateA and launch a traceroute from there.

## Testing in stages

It is often useful in debugging to test things one at a time:

- disable IPsec entirely, for example by turning it off with `chkconfig(8)`, and make sure routing works
- Once that works, try a manually keyed connection. This does not require key negotiation between Pluto and the key daemon on the other end.
- Once that works, try automatically keyed connections
- Once IPsec works, add packet compression
- Once everything seems to work, try stress tests with large transfers, many connections, frequent re-keying, ...

FreeS/WAN releases are tested for all of these, so you can be reasonably certain they *can* do them all. Of course, that does not mean they *will* on the first try, especially if you have some unusual configuration.

The rest of this section gives information on diagnosing the problem when each of the above steps fails.

## Manually keyed connections don't work

Suspect one of:

- mis-configuration of IPsec system in the `/etc/ipsec.conf` file  
common errors are incorrect interface or next hop information
- mis-configuration of manual connection in the `/etc/ipsec.conf` file
- routing problems causing IPsec packets to be lost
- bugs in KLIPS
- mismatch between the transforms we support and those another IPsec implementation offers.

## One manual connection works, but second one fails

This is a fairly common problem when attempting to configure multiple manually keyed connections from a single gateway.

Each connection must be identified by a unique SPI value. For automatic connections, these values are assigned automatically. For manual connections, you must set them with `spi=` statements in `ipsec.conf(5)`.

Each manual connection must have a unique SPI value in the range 0x100 to 0x999. Two or more with the same value will fail. For details, see our doc section Using manual keying in production and the man page `ipsec.conf(5)`.

## Manual connections work, but automatic keying doesn't

The most common reason for this behaviour is a firewall dropping the UDP port 500 packets used in key negotiation.

Other possibilities:

- mis-configuration of auto connection in the `/etc/ipsec.conf` file.

One common configuration error is forgetting that you need *auto=add* to load the connection description on the receiving end so it recognises the connection when the other end asks for it.

- error in shared secret in `/etc/ipsec.secrets`
- one gateway lacks a route to the other so Pluto's UDP packets are lost
- bugs in Pluto
- incompatibilities between Pluto's IKE implementation and the IKE at the other end of the tunnel.

Some possible problems are discussed in our interoperation document.

## IPsec works, but connections using compression fail

When we first added compression, we saw some problems:

- compatibility issues with other implementations. We followed the RFCs and omitted some extra material that many compression libraries add by default. Some other implementations left the extras in
- bugs in assembler compression routines on non-Intel CPUs. The workaround is to use C code instead of possibly problematic assembler.

We have not seen either problem in some time (at least six months as I write in March 2002), but if you have some unusual configuration then you may see them.

## Small packets work, but large transfers fail

If tests with `ping(1)` and a small packet size succeed, but tests or transfers with larger packet sizes fail, suspect problems with packet fragmentation and perhaps path MTU discovery.

Our troubleshooting document covers these problems. Information on the underlying mechanism is in our background document.

## Subnet-to-subnet works, but tests from the gateways don't

This is described under I cannot ping... above.

## Compilation problems

### gmp.h: No such file or directory

Pluto needs the GMP (GNU

*Multi-Precision*) library for the large integer calculations it uses in public key cryptography. This error message indicates a failure to find the library. You must install it before Pluto will compile.

The GMP library is included in most Linux distributions. Typically, there are two RPMs, libgmp and libgmp-devel, You need to *install both*, either from your distribution CDs or from your vendor's web site.

On Debian, a mailing list message reports that the command to give is *apt-get install gmp2*.

For more information and the latest version, see the GMP home page.

### ... virtual memory exhausted

We have had several reports of this message appearing, all on SPARC Linux. Here is a mailing message on a solution:

```
> ipsec_shal.c: In function `SHA1Transform':  
> ipsec_shal.c:95: virtual memory exhausted
```

I'm seeing exactly the same problem on an Ultra with 256MB ram and 500 MB swap. Except I am compiling version 1.5 and its Red Hat 6.2.

I can get around this by using -O instead of -O2 for the optimization level. So it is probably a bug in the optimizer on the sparc compiler. I'll try and chase this down on the sparc lists.

## Interpreting error messages

### route-client (or host) exited with status 7

Here is a discussion of this error from FreeS/WAN "listress" (mailing list tech support person) Claudia Schmeing. The "FAQ on the network unreachable error" which she refers to is the next question below.

```
> I reached the point where the two boxes (both on dial-up connections, but  
> treated as static IPs by getting the IP and editing ipsec.conf after the  
> connection is established) to the point where they exchange some info, but I  
> get an error like "route-client command exited with status 7 \n internal  
> error".  
> Where can I find a description of this error?
```

In general, if the FAQ doesn't cover it, you can search the mailing list archives - I like to use <http://www.sandelman.ottawa.on.ca/linux-ipsec/> but you can see doc/mail.html for different archive formats.

Your error comes from the \_updown script, which performs some routing and firewall functions to help Linux FreeS/WAN. More info is available at doc/firewall.html and man ipsec.conf. Its routing is integral to the health of Linux FreeS/WAN; it also provides facility to insert custom firewall rules to be executed when you create or destroy a connection.

Yours is, of course, a routing error. You can be fairly sure the routing machinery is saying "network is unreachable". There's a FAQ on the "network is unreachable" error, but more information is available now; read on.

## Introduction to FreeS/WAN

If your `_updown` script is recent (for example if it shipped with Linux FreeS/WAN 1.91), you will see another debugging line in your logs that looks something like this:

```
> output: /usr/local/lib/ipsec/_updown: `route add -net 128.174.253.83
> netmask 255.255.255.255 dev ipsec0 gw 66.92.93.161' failed
```

This is, of course, the system `route` command that exited with status 7, (ie. failed). Man `route` for details. Seeing the command typed out yields more information. If your `_updown` script is older, you may wish to update it to show the command explicitly.

Three parameters fed to the `route` command: `net`, `netmask` and `gw` [gateway] are derived from things you've put in `ipsec.conf`.

`Net` and `netmask` are derived from the peer's IP and mask. In more detail:

You may see a routing error when routing to a client (ie. subnet), or to a host (IPSec gateway or freestanding host; a box that does IPSec for itself). In `_updown`, the "route-client" section is responsible to set up the route for IPSec'd (usually, read 'tunneled') packets headed to a peer subnet. Similarly, `route-host` routes IPSec'd packets to a peer host or IPSec gateway.

When routing to a 'client', `net` and `netmask` are `ipsec.conf`'s `left-` or `rightsubnet` (whichever is not local). Similarly, when routing to a 'host' the `net` is left or right. Host `netmask` is always `/32`, indicating a single machine.

`Gw` is `nexthop`'s value. Again, the value in question is `left-` or `rightnexthop`, whichever is local. Where `left/right` or `left-/rightnexthop` has the special value `%defaultroute` (described in `man ipsec.conf`), `gw` will automatically get the value of the next hop on the default route.

Q: "What's a `nexthop` and why do I need one?"

A: '`nexthop`' is a routing kluge; its value is the next hop away from the machine that's doing IPSec, and toward your IPSec peer. You need it to get the processed packets out of the local system and onto the wire. While we often route other packets through the machine that's now doing IPSec, and are done with it, this does not suffice here. After packets are processed with IPSec, this machine needs to know where they go next. Of course using the 'IPSec gateway' as their routing gateway would cause an infinite loop! [To visualize this, see the packet flow diagram at `doc/firewall.html`.] To avoid this, we route packets through the next hop down their projected path.

Now that you know the background, consider:

1. Did you test routing between the gateways in the absence of Linux FreeS/WAN, as recommended? You need to ensure the two machines that will be running Linux FreeS/WAN can route to one another before trying to make a secure connection.
2. Is there anything obviously wrong with the sense of your `route` command?

Normally, this problem is caused by an incorrect local `nexthop` parameter. Check out the use of `%defaultroute`, described in `man ipsec.conf`. This is a simple way to set `nexthop` for most people. To figure `nexthop` out by hand, `traceroute` in-the-clear to your IPSec peer. `Nexthop` is the `traceroute`'s first hop after your IPSec gateway.

## SIOCADDRT:Network is unreachable

This message is not from FreeS/WAN, but from the Linux IP stack itself. That stack is seeing packets it has no route for, either because your routing was broken before FreeS/WAN started or because FreeS/WAN's changes broke it.

Here is a message from Claudia suggesting ways to diagnose and fix such problems:

```
You write,  
> I have correctly installed freeswan-1.8 on RH7.0 kernel 2.2.17, but when  
> I setup a VPN connection with the other machine(RH5.2 Kernel 2.0.36  
> freeswan-1.0, it works well.) it told me that  
> "SIOCADDRT:Network is unreachable"! But the network connection is no  
> problem.
```

Often this error is the result of a misconfiguration.

Be sure that you can route successfully in the absence of Linux FreeS/WAN. (You say this is no problem, so proceed to the next step.)

Use a custom copy of the default updownscript. Do not change the route commands, but add a diagnostic message revealing the exact text of the route command. Is there a problem with the sense of the route command that you can see? If so, then re-examine those ipsec.conf settings that are being sent to the route command.

You may wish to use the ipsec auto --route and --unroute commands to troubleshoot the problem. See man ipsec\_auto for details.

Since the above message was written, we have modified the updown script to provide a better diagnostic for this problem. Check */var/log/messages*.

See also the FAQ question [route-client \(or host\) exited with status 7](#).

## ipsec\_setup: modprobe: Can't locate module ipsec

## ipsec\_setup: Fatal error, kernel appears to lack KLIPS

These messages indicate an installation failure. The kernel you are running does not contain the KLIPS (kernel IPsec) code.

Note that the "modprobe: Can't locate module ipsec" message appears even if you are not using modules. If there is no KLIPS in your kernel, FreeS/WAN tries to load it as a module. If that fails, you get this message.

Commands you can quickly try are:

*uname -a*

to get details, including compilation date and time, of the currently running kernel

*ls /*

*ls /boot*

to ensure a new kernel is where it should be. If kernel compilation puts it in */* but *lilo* wants it in */boot*, then you should uncomment the *INSTALL\_PATH=/boot* line in the kernel *Makefile*.

*more /etc/lilo.conf*

to see that *lilo* has correct information

*lilo*

to ensure that information in */etc/lilo.conf* has been transferred to the boot sector

If those don't find the problem, you have to go back and check through the install procedure to see what was missed.

Here is one of Claudia's messages on the topic:

```
> I tried to install freeswan 1.8 on my mandrake 7.2 test box. ...
```

```
> It does show version and some output for whack.
```

Yes, because the Pluto (daemon) part of ipsec is installed correctly, but as we see below the kernel portion is not.

```
> However, I get the following from /var/log/messages:
```

```
>
```

```
> Mar 11 22:11:55 pavillion ipsec_setup: Starting FreeS/WAN IPsec 1.8...
```

```
> Mar 11 22:12:02 pavillion ipsec_setup: modprobe: Can't locate module ipsec
```

```
> Mar 11 22:12:02 pavillion ipsec_setup: Fatal error, kernel appears to lack
```

```
> KLIPS.
```

This is your problem. You have not successfully installed a kernel with IPSec machinery in it.

Did you build Linux FreeS/WAN as a module? If so, you need to ensure that your new module has been installed in the directory where your kernel loader normally finds your modules. If not, you need to ensure that the new IPSec-enabled kernel is being loaded correctly.

See also doc/install.html, and INSTALL in the distro.

## ipsec\_setup: ... failure to fetch key for ... from DNS

Quoting Henry:

Note that by default, FreeS/WAN is now set up to

- (a) authenticate with RSA keys, and

- (b) fetch the public key of the far end from DNS.

Explicit attention to *ipsec.conf* will be needed if you want to do something different.

and Claudia, responding to the same user:

You write,

```
> My current setup in ipsec.conf is leftrsasigkey=%dns I have
> commented this and authby=rsasig out. I am able to get ipsec running,
> but what I find is that the documentation only specifies for %dns are
> there any other values that can be placed in this variable other than
> %dns and the key? I am also assuming that this is where I would place
> my public key for the left and right side as well is this correct?
```

Valid values for *authby=* are *rsasig* and *secret*, which entail authentication by RSA signature or by shared secret, respectively. Because you have commented *authby=rsasig* out, you are using the default value of *authby=secret*.

When using RSA signatures, there are two ways to get the public key for the

## Introduction to FreeS/WAN

IPSec peer: either copy it directly into `*rsasigkey=` in `ipsec.conf`, or fetch it from dns. The magic value `%dns` for `*rsasigkey` parameters says to try to fetch the peer's key from dns.

For any parameters, you may find their significance and special values in `man ipsec.conf`. If you are setting up keys or secrets, be sure also to reference `man ipsec.secrets`.

### **ipsec\_setup: ... interfaces ... and ... share address ...**

This is a fatal error. FreeS/WAN cannot cope with two or more interfaces using the same IP address. You must re-configure to avoid this.

A mailing list message on the topic from Pluto developer Hugh Redelmeier:

```
| I'm trying to get freeswan working between two machine where one has a ppp
| interface.
| I've already succeeded with two machines with ethernet ports but the ppp
| interface is causing me problems.
| basically when I run ipsec start i get
| ipsec_setup: Starting FreeS/WAN IPsec 1.7...
| ipsec_setup: 003 IP interfaces pppl and ppp0 share address 192.168.0.10!
| ipsec_setup: 003 IP interfaces pppl and ppp2 share address 192.168.0.10!
| ipsec_setup: 003 IP interfaces ppp0 and ppp2 share address 192.168.0.10!
| ipsec_setup: 003 no public interfaces found
|
| followed by lots of cannot work out interface for connection messages
|
| now I can specify the interface in ipsec.conf to be ppp0 , but this does
| not affect the above behaviour. A quick look in server.c indicates that the
| interfaces value is not used but some sort of raw detect happens.
|
| I guess I could prevent the formation of the extra ppp interfaces or
| allocate them different ip but I'd rather not. if at all possible. Any
| suggestions please.
```

Pluto won't touch an interface that shares an IP address with another. This will eventually change, but it probably won't happen soon.

For now, you will have to give the pppl and ppp2 different addresses.

### **ipsec\_setup: Cannot adjust kernel flags**

A mailing list message from technical lead Henry Spencer:

```
> When FreeS/WAN IPsec 1.7 is starting on my 2.0.38 Linux kernel the following
> error message is generated:
> ipsec_setup: Cannot adjust kernel flags, no /proc/sys/net/ipsec directory!
> What is supposed to create this directory and how can I fix this problem?
```

I think that directory is a 2.2ism, although I'm not certain (I don't have a 2.0.xx system handy any more for testing). Without it, some of the `ipsec.conf` config-setup flags won't work, but otherwise things should function.

You also need to enable the `/proc` filesystem in your kernel configuration for these operations to work.

## Message numbers (MI3, QR1, et cetera) in Pluto messages

Pluto messages often indicate where Pluto is in the IKE protocols. The letters indicate *M*ain mode or *Q*uick mode and *I*nitiator or *R*esponder. The numerals are message sequence numbers. For more detail, see our [IPsec section](#).

## Connection names in Pluto error messages

From Pluto programmer Hugh Redelmeier:

```
| Jan 17 16:21:10 remus Pluto[13631]: "jumble" #1: responding to Main Mode from Road Warrior 130.  
| Jan 17 16:21:11 remus Pluto[13631]: "jumble" #1: no suitable connection for peer @banshee.witts  
|  
|       The connection "jumble" has nothing to do with the incoming  
| connection requests, which were meant for the connection "banshee".
```

You are right. The message tells you which Connection Pluto is currently using, which need not be the right one. It need not be the right one now for the negotiation to eventually succeed! This is described in `ipsec_pluto(8)` in the section "Road Warrior Support".

There are two times when Pluto will consider switching Connections for a state object. Both are in response to receiving ID payloads (one in Phase 1 / Main Mode and one in Phase 2 / Quick Mode). The second is not unique to Road Warriors. In fact, neither is the first any more (two connections for the same pair of hosts could differ in Phase 1 ID payload; probably nobody else has tried this).

## Pluto: ... can't orient connection

Older versions of FreeS/WAN used this message. The same error now gives the "we have no ipsecN ..." error described just below.

## ... we have no ipsecN interface for either end of this connection

Your tunnel has no IP address which matches the IP address of any of the available IPsec interfaces. Either you've misconfigured the connection, or you need to define an appropriate IPsec interface connection. *interfaces=%defaultroute* works in many cases.

A longer story: Pluto needs to know whether it is running on the machine which the connection description calls *left* or on *right*. It figures that out by:

- looking at the interfaces given in *interfaces=* lines in the *config setup* section
- discovering the IP addresses for those interfaces
- searching for a match between those addresses and the ones given in *left=* or *right=* lines.

Normally a match is found. Then Pluto knows where it is and can set up other things (for example, if it is *left*) using parameters such as *leftsubnet* and *leftnexthop*, and sending its outgoing packets to *right*.

If no match is found, it emits the above error message.

## Pluto: ... no connection is known

This error message occurs when a remote system attempts to negotiate a connection and Pluto does not have a connection description that matches what the remote system has requested. The most common cause is a configuration error on one end or the other.

Parameters involved in this match are *left*, *right*, *leftsubnet* and *rightsubnet*.

**The match must be exact.** For example, if your left subnet is a.b.c.0/24 then neither a single machine in that net nor a smaller subnet such as a.b.c.64/26 will be considered a match.

The message can also occur when an appropriate description exists but Pluto has not loaded it. Use an *auto=add* statement in the connection description, or an *ipsec auto --add <conn\_name>* command, to correct this.

An explanation from the Pluto developer:

```
| Jul 12 15:00:22 sohar58 Pluto[574]: "corp_road" #2: cannot respond to IPsec
| SA request because no connection is known for
| 216.112.83.112/32===216.112.83.112...216.67.25.118
```

This is the first message from the Pluto log showing a problem. It means that PGPnet is trying to negotiate a set of SAs with this topology:

```
216.112.83.112/32===216.112.83.112...216.67.25.118
^^^^^^^^^^^^^^^^  ^^^^^^^^^^^^^^^^^  ^^^^^^^^^^^^^^^^^
client on our side  our host           PGPnet host, no client
```

None of the conns you showed look like this.

Use

```
    ipsec auto --status
to see a snapshot of what connections are in pluto, what
negotiations are going on, and what SAs are established.
```

The leftsubnet= (client) in your conn is 216.112.83.64/26. It must exactly match what pluto is looking for, and it does not.

## Pluto: ... no suitable connection ...

This is similar to the no connection known error, but occurs at a different point in Pluto processing.

Here is one of Claudia's messages explaining the problem:

You write,

```
> What could be the reason of the following error?
> "no suitable connection for peer '@xforce'"
```

When a connection is initiated by the peer, Pluto must choose which entry in the conf file best matches the incoming connection. A preliminary choice is made on the basis of source and destination IPs, since that information is available at that time.

A payload containing an ID arrives later in the negotiation. Based on this

## Introduction to FreeS/WAN

id and the \*id= parameters, Pluto refines its conn selection. ...

The message "no suitable connection" indicates that in this refining step, Pluto does not find a connection that matches that ID.

Please see "Selecting a connection when responding" in man ipsec\_pluto for more details.

See also [Connection names in Pluto error messages](#).

## Pluto: ... no connection has been authorized

Here is one of Claudia's messages discussing this problem:

You write,

```
> May 22 10:46:31 debian Pluto[25834]: packet from x.y.z.p:10014:
> initial Main Mode message from x.y.z.p:10014
    but no connection has been authorized
```

This error occurs early in the connection negotiation process, at the first step of IKE negotiation (Main Mode), which is itself the first of two negotiation phases involved in creating an IPSec connection.

Here, Linux FreeS/WAN receives a packet from a potential peer, which requests that they begin discussing a connection.

The "no connection has been authorized" means that there is no connection description in Linux FreeS/WAN's internal database that can be used to link your ipsec interface with that peer.

"But of course I configured that connection!"

It may be that the appropriate connection description exists in ipsec.conf but has not been added to the database with ipsec auto --add myconn or the auto=add method. Or, the connection description may be misconfigured.

The only parameters that are relevant in this decision are left= and right= . Local and remote ports are also taken into account -- we see that the port is printed in the message above -- but there is no way to control these in ipsec.conf.

Failure at "no connection has been authorized" is similar to the "no connection is known for..." error in the FAQ, and the "no suitable connection" error described in the snapshot's FAQ. In all three cases, Linux FreeS/WAN is trying to match parameters received in the negotiation with the connection description in the local config file.

As it receives more information, its matches take more parameters into account, and become more precise: first the pair of potential peers, then the peer IDs, then the endpoints (including any subnets).

The "no suitable connection for peer \*" occurs toward the end of IKE (Main Mode) negotiation, when the IDs are matched.

"no connection is known for a/b==c...d" is seen at the beginning of IPSec (Quick Mode, phase 2) negotiation, when the connections are matched using left, right, and any information about the subnets.

## **Pluto: ... OAKLEY\_DES\_CBC is not supported.**

This message occurs when the other system attempts to negotiate a connection using single DES, which we do not support because it is insecure.

Our interoperation document has suggestions for how to deal with systems that attempt to use single DES.

## **Pluto: ... no acceptable transform**

This message means that the other gateway has made a proposal for connection parameters, but nothing they proposed is acceptable to Pluto. Possible causes include:

- misconfiguration on either end
- policy incompatibilities, for example we require encrypted connections but they are trying to create one with just authentication
- interoperation problems, for example they offer only single DES and FreeS/WAN does not support that. See discussion in our interoperation document.

A more detailed explanation, from Pluto programmer Hugh Redelmeier:

Background:

When one IKE system (for example, Pluto) is negotiating with another to create an SA, the Initiator proposes a bunch of choices and the Responder replies with one that it has selected.

The structure of the choices is fairly complicated. An SA payload contains a list of lists of "Proposals". The outer list is a set of choices: the selection must be from one element of this list.

Each of these elements is a list of Proposals. A selection must be made from each of the elements of the inner list. In other words, *\*all\** of them apply (that is how, for example, both AH and ESP can apply at once).

Within each of these Proposals is a list of Transforms. For each Proposal selected, one Transform must be selected (in other words, each Proposal provides a choice of Transforms).

Each Transform is made up of a list of Attributes describing, well, attributes. Such as lifetime of the SA. Such as algorithm to be used. All the Attributes apply to a Transform.

You will have noticed a pattern here: layers alternate between being disjunctions ("or") and conjunctions ("and").

For Phase 1 / Main Mode (negotiating an ISAKMP SA), this structure is cut back. There must be exactly one Proposal. So this degenerates to a list of Transforms, one of which must be chosen.

In your case, no proposal was considered acceptable to Pluto (the Responder). So negotiation ceased. Pluto logs the reason it rejects each Transform. So look back in the log to see what is going wrong.

## rsasigkey dumps core

A comment on this error from Henry:

On Fri, 29 Jun 2001, Rodrigo Gruppelli wrote:

```
> ...Well, it seem that there's  
> another problem with it. When I try to generate a pair of RSA keys,  
> rsasigkey cores dump...
```

\*That\* is a neon sign flashing "GMP LIBRARY IS BROKEN". Rsasigkey calls GMP a lot, and our own library a little bit, and that's very nearly all it does. Barring bugs in its code or our library -- which have happened, but not very often -- a problem in rsasigkey is a problem in GMP.

See the next question for how to deal with GMP errors.

## !Pluto failure!: ... exited with ... signal 4

Pluto has died. Signal 4 is SIGILL, illegal instruction.

The most likely cause is that your GMP (GNU multi-precision) library is compiled for a different processor than what you are running on. Pluto uses that library for its public key calculations.

Try getting the GMP sources and recompile for your processor type. Most Linux distributions will include this source, or you can download it from the GMP home page.

## ECONNREFUSED error message

From John Denker, on the mailing list:

- 1) The log message  
some IKE message we sent has been rejected with  
ECONNREFUSED (kernel supplied no details)  
is much more suitable than the previous version. Thanks.
- 2) Minor suggestion for further improvement: it might be worth mentioning that the command  
tcpdump -i eth1 icmp[0] != 8 and icmp[0] != 0  
is useful for tracking down the details in question. We shouldn't expect all IPsec users to figure that out on their own. The log message might even provide a hint as to where to look in the docs.

Reply From Pluto developer Hugh Redelmeier

Good idea.

I've added a bit pluto(8)'s BUGS section along these lines.  
I didn't have the heart to lengthen this message.

## klips\_debug: ... no eroute!

This message means KLIPS has received a packet for which no IPsec tunnel has been defined.

## Introduction to FreeS/WAN

Here is a more detailed discussion from the team's tech support person Claudia Schmeing, responding to a query on the mailing list:

```
> Why ipsec reports no eroute! ??? IP Masq... is disabled.
```

In general, more information is required so that people on the list may give you informed input. See doc/prob.report.

The document she refers to has since been replaced by a section of the troubleshooting document.

However, I can make some general comments on this type of error.

This error usually looks something like this (clipped from an archived message):

```
> ttl:64 proto:1 chk:45459 saddr:192.168.1.2 daddr:192.168.100.1
> ... klips_debug:ipsec_findroute: 192.168.1.2->192.168.100.1
> ... klips_debug:rj_match: * See if we match exactly as a host destination
> ... klips_debug:rj_match: ** try to match a leaf, t=0xc1a260b0
> ... klips_debug:rj_match: *** start searching up the tree, t=0xc1a260b0
> ... klips_debug:rj_match: **** t=0xc1a260c8
> ... klips_debug:rj_match: **** t=0xc1fe5960
> ... klips_debug:rj_match: ***** not found.
> ... klips_debug:ipsec_tunnel_start_xmit: Original head/tailroom: 2, 28
> ... klips_debug:ipsec_tunnel_start_xmit: no eroute!: ts=47.3030, dropping.
```

What does this mean?

- -----

"eroute" stands for "extended route", and is a special type of route internal to Linux FreeS/WAN. For more information about this type of route, see the section of man ipsec\_auto on ipsec auto --route.

"no eroute!" here means, roughly, that Linux FreeS/WAN cannot find an appropriate tunnel that should have delivered this packet. Linux FreeS/WAN therefore drops the packet, with the message "no eroute! ... dropping", on the assumption that this packet is not a legitimate transmission through a properly constructed tunnel.

How does this situation come about?

- -----

Linux FreeS/WAN has a number of connection descriptions defined in ipsec.conf. These must be successfully brought "up" to form actual tunnels. (see doc/setup.html's step 15, man ipsec.conf and man ipsec\_auto for details).

Such connections are often specific to the endpoints' IPs. However, in some cases they may be more general, for example in the case of Road Warriors where left or right is the special value %any.

When Linux FreeS/WAN receives a packet, it verifies that the packet has come through a legitimate channel, by checking that there is an appropriate tunnel through which this packet might legitimately have arrived. This is the process we see above.

First, it checks for an eroute that exactly matches the packet. In the example above, we see it checking for a route that begins at 192.168.1.2 and ends at 192.168.100.1. This search favours the most specific match that

## Introduction to FreeS/WAN

would apply to the route between these IPs. So, if there is a connection description exactly matching these IPs, the search will end there. If not, the code will search for a more general description matching the IPs. If there is no match, either specific or general, the packet will be dropped, as we see, above.

Unless you are working with Road Warriors, only the first, specific part of the matching process is likely to be relevant to you.

"But I defined the tunnel, and it came up, why do I have this error?"

-----

One of the most common causes of this error is failure to specify enough connection descriptions to cover all needed tunnels between any two gateways and their respective subnets. As you have noticed, troubleshooting this error may be complicated by the use of IP Masq. However, this error is not limited to cases where IP Masq is used.

See [doc/configuration.html#multitunnel](#) for a detailed example of the solution to this type of problem.

The documentation section she refers to is now [here](#).

### ... trouble writing to /dev/ipsec ... SA already in use

This error message occurs when two manual connections are set up with the same SPI value.

See the FAQ for [One manual connection works, but second one fails](#).

### ... ignoring ... payload

This message is harmless. The IKE protocol provides for a number of optional messages types:

- delete SA
- initial contact
- vendor ID
- ...

An implementation is never required to send these, but they are allowed to. The receiver is not required to do anything with them. FreeS/WAN ignores them, but notifies you via the logs.

For the "ignoring delete SA Payload" message, see also our discussion of cleaning up [dead tunnels](#).

### unknown parameter name "rightcert"

This message can appear when you've upgraded an X.509-enabled Linux FreeS/WAN with a vanilla Linux FreeS/WAN. To use your X.509 configs you will need to overwrite the new install with [Super FreeS/WAN](#), or add the [X.509 patch](#) by hand.

## Why don't you restrict the mailing lists to reduce spam?

As a matter of policy, some of our [mailing lists](#) need to be open to non-subscribers. Project management feel

## Introduction to FreeS/WAN

strongly that maintaining this openness is more important than blocking spam.

- Users should be able to get help or report bugs without subscribing.
- Even a user who is subscribed may not have access to his or her subscribed account when he or she needs help, miles from home base in the middle of setting up a client's gateway.
- There is arguably a legal requirement for this policy. A US resident or citizen could be charged under munitions export laws for providing technical assistance to a foreign cryptographic project. Such a charge would be more easily defended if the discussion takes place in public, on an open list.

This has been discussed several times at some length on the list. See the [list archives](#). Bringing the topic up again is unlikely to be useful. Please don't. Or at the very least, please don't without reading the archives and being certain that whatever you are about to suggest has not yet been discussed.

Project technical lead Henry Spencer summarised one discussion:

For the third and last time: this list *\*will\* \*not\** do address-based filtering. This is a policy decision, not an implementation problem. The decision is final, and is not open to discussion. This needs to be communicated better to people, and steps are being taken to do that.

Adding this FAQ section is one of the steps he refers to.

You have various options other than just putting up with the spam, filtering it yourself, or unsubscribing:

- subscribe only to one or both of our lists with restricted posting rules:
  - ♦ [briefs](#), weekly list summaries
  - ♦ [announce](#), project-related announcements
- read the other lists via the [archives](#)

A number of tools are available to filter mail.

- Many mail readers include some filtering capability.
- Many Linux distributions include [procmail\(8\)](#) for server-side filtering.
- The [Spam Bouncer](#) is a set of procmail(8) filters designed to combat spam.
- Roaring Penguin have a [MIME defanger](#) that removes potentially dangerous attachments.

If you use your ISP's mail server rather than running your own, consider suggesting to the ISP that they tag suspected spam as [this ISP](#) does. They could just refuse mail from dubious sources, but that is tricky and runs some risk of losing valuable mail or senselessly annoying senders and their admins. However, they can safely tag and deliver dubious mail. The tags can greatly assist your filtering.

For information on tracking down spammers, see these [HowTos](#), or the [Sputum](#) site. Sputum have a Linux anti-spam screensaver available for download.

Here is a more detailed message from Henry:

On Mon, 15 Jan 2001, Jay Vaughan wrote:

```
> I know I'm flogging a dead horse here, but I'm curious as to the reasons for  
> an aversion for a subscriber-only mailing list?
```

Once again: for legal reasons, it is important that discussions of these things be held in a public place -- the list -- and we do not want to force people to subscribe to the list just to ask one question, because

## Introduction to FreeS/WAN

that may be more than merely inconvenient for them. There are also real difficulties with people who are temporarily forced to use alternate addresses; that is precisely the time when they may be most in need of help, yet a subscribers-only policy shuts them out.

These issues do not apply to most mailing lists, but for a list that is (necessarily) the primary user support route for a crypto package, they are very important. This is *\*not\** an ordinary mailing list; it has to function under awkward constraints that make various simplistic solutions inapplicable or undesirable.

> We're *\*ALL\** sick of hearing about list management problems, not just you  
> old-timers, so why don't you DO SOMETHING EFFECTIVE ABOUT IT...

Because it's a lot harder than it looks, and many existing "solutions" have problems when examined closely.

> A suggestion for you, based on 10 years of experience with management of my  
> own mailing lists would be to use mailman, which includes pretty much every  
> feature under the sun that you guys need and want, plus some. The URL for  
> mailman...

I assure you, we're aware of mailman. Along with a whole bunch of others, including some you almost certainly have never heard of (I hadn't!).

> As for the argument that the list shouldn't be configured to enforce  
> subscription - I contend that it *\*SHOULD\** AT LEAST require manual address  
> verification in order for posts to be redirected.

You do realize, I hope, that interposing such a manual step might cause your government to decide that this is not truly a public forum, and thus you could go to jail if you don't get approval from them before mailing to it? If you think this sounds irrational, your government is noted for making irrational decisions in this area; we can't assume that they will suddenly start being sensible. See above about awkward constraints. You may be willing to take the risk, but we can't, in good conscience, insist that all users with problems do so.

Henry Spencer  
henry@spsystems.net

and a message on the topic from project leader John Gilmore:

Subject: Re: The linux-ipsec list's topic  
Date: Sat, 30 Dec 2000  
From: John Gilmore <gnu@toad.com>

I'll post this single message, once only, in this discussion, and then not burden the list with any further off-topic messages. I encourage everyone on the list to restrain themself from posting ANY off-topic messages to the linux-ipsec list.

The topic of the linux-ipsec mailing list is the FreeS/WAN software.

I frequently see "discussions about spam on a list" overwhelm the volume of "actual spam" on a list. BOTH kinds of messages are off-topic messages. Twenty anti-spam messages take just as long to detect and discard as twenty spam messages.

The Linux-ipsec list encourages on-topic messages from people who have not joined the list itself. We will not censor messages to the list based on where they originate, or what return address they contain.

... trouble writing to /dev/ipsec ... SA already in use

## Introduction to FreeS/WAN

In other words, non-subscribers ARE allowed to post, and this will not change. My own valid contributions have been rejected out-of-hand by too many other mailing lists for me to want to impose that censorship on anybody else's contributions. And every day I see the damage that anti-spam zeal is causing in many other ways; that zeal is far more damaging to the culture of the Internet than the nuisance of spam.

In general, it is the responsibility of recipients to filter, prioritize, or otherwise manage the handling of email that comes to them. It is not the responsibility of the rest of the Internet community to refrain from sending messages to recipients that they might not want to see. If your software infrastructure for managing your incoming email is insufficient, then improve it. If you think the signal-to-noise ratio on linux-ipsec is too poor, then please unsubscribe. But don't further increase the noise by posting to the linux-ipsec list about those topics.

John Gilmore  
founder & sponsor, FreeS/WAN project

# FreeS/WAN manual pages

The various components of Linux FreeS/WAN are of course documented in standard Unix manual pages, accessible via the `man(1)` command.

Links here take you to an HTML version of the man pages.

## Files

[ipsec.conf\(5\)](#)

IPsec configuration and connections

[ipsec.secrets\(5\)](#)

secrets for IKE authentication, either pre-shared keys or RSA private keys

These files are also discussed in the [configuration](#) section.

## Commands

Many users will never give most of the FreeS/WAN commands directly. Configure the files listed above correctly and everything should be automatic.

The exceptions are commands for manipulating the [RSA](#) keys used in Pluto authentication:

[ipsec\\_rasigkey\(8\)](#)

generate keys

[ipsec\\_newhostkey\(8\)](#)

generate keys in a convenient format

[ipsec\\_showhostkey\(8\)](#)

extract [RSA](#) keys from [ipsec.secrets\(5\)](#) (or optionally, another file) and format them for insertion in [ipsec.conf\(5\)](#) or in DNS records

Note that:

- These keys are for ***authentication only***. They are ***not secure for encryption***.
- The utility uses `random(4)` as a source of [random numbers](#). This may block for some time if there is not enough activity on the machine to provide the required entropy. You may want to give it some bogus activity such as random mouse movements or some command such as `du /usr > /dev/null &`.

The following commands are fairly likely to be used, if only for testing and status checks:

[ipsec\(8\)](#)

invoke IPsec utilities

[ipsec\\_setup\(8\)](#)

control IPsec subsystem

[ipsec\\_auto\(8\)](#)

control automatically-keyed IPsec connections

[ipsec\\_manual\(8\)](#)

take manually-keyed IPsec connections up and down

*ipsec ranbits(8)*

generate random bits in ASCII form

*ipsec look(8)*

show minimal debugging information

*ipsec barf(8)*

spew out collected IPsec debugging information

The lower-level utilities listed below are normally invoked via scripts listed above, but they can also be used directly when required.

*ipsec eroute(8)*

manipulate IPsec extended routing tables

*ipsec klipsdebug(8)*

set Klips (kernel IPsec support) debug features and level

*ipsec pluto(8)*

IPsec IKE keying daemon

*ipsec spi(8)*

manage IPsec Security Associations

*ipsec spigrp(8)*

group/ungroup IPsec Security Associations

*ipsec tncfg(8)*

associate IPsec virtual interface with real interface

*ipsec whack(8)*

control interface for IPsec keying daemon

## Library routines

*ipsec atoaddr(3)*

*ipsec addrtoa(3)*

convert Internet addresses to and from ASCII

*ipsec atosubnet(3)*

*ipsec subnettoa(3)*

convert subnet/mask ASCII form to and from addresses

*ipsec atoasr(3)*

convert ASCII to Internet address, subnet, or range

*ipsec rangetoa(3)*

convert Internet address range to ASCII

*ipsec\_atodata(3)*

*ipsec\_datatoa(3)*

convert binary data from and to ASCII formats

*ipsec atosa(3)*

*ipsec satoa(3)*

convert IPsec Security Association IDs to and from ASCII

*ipsec atoul(3)*

*ipsec ultoa(3)*

convert unsigned-long numbers to and from ASCII

*ipsec goodmask(3)*

is this Internet subnet mask a valid one?

*ipsec masktobits(3)*

convert Internet subnet mask to bit count

*ipsec bitstomask(3)*

convert bit count to Internet subnet mask

*ipsec optionsfrom(3)*

read additional ``command-line" options from file

*ipsec subnetof(3)*

given Internet address and subnet mask, return subnet number

*ipsec hostof(3)*

given Internet address and subnet mask, return host part

*ipsec broadcastof(3)*

given Internet address and subnet mask, return broadcast address

# FreeS/WAN and firewalls

FreeS/WAN, or other IPsec implementations, frequently run on gateway machines, the same machines running firewall or packet filtering code. This document discusses the relation between the two.

The firewall code in 2.4 and later kernels is called Netfilter. The user-space utility to manage a firewall is iptables(8). See the [netfilter/iptables web site](#) for details.

## Filtering rules for IPsec packets

The basic constraint is that *an IPsec gateway must have packet filters that allow IPsec packets*, at least when talking to other IPsec gateways:

- UDP port 500 for IKE negotiations
- protocol 50 if you use ESP encryption and/or authentication (the typical case)
- protocol 51 if you use AH packet-level authentication

Your gateway and the other IPsec gateways it communicates with must be able to exchange these packets for IPsec to work. Firewall rules must allow UDP 500 and at least one of AH or ESP on the interface that communicates with the other gateway.

For nearly all FreeS/WAN applications, you must allow UDP port 500 and the ESP protocol.

There are two ways to set this up:

*easier but less flexible*

Just set up your firewall scripts at boot time to allow IPsec packets to and from your gateway. Let FreeS/WAN reject any bogus packets.

*more work, giving you more precise control*

Have the ipsec\_pluto(8) daemon call scripts to adjust firewall rules dynamically as required. This is done by naming the scripts in the ipsec.conf(5) variables *prepluto=*, *postpluto=*, *leftupdown=* and *rightupdown=*.

Both methods are described in more detail below.

## Firewall configuration at boot

It is possible to set up both firewalling and IPsec with appropriate scripts at boot and then not use *leftupdown=* and *rightupdown=*, or use them only for simple up and down operations.

Basically, the technique is

- allow IPsec packets (typically, IKE on UDP port 500 plus ESP, protocol 50)
  - ◆ incoming, if the destination address is your gateway (and optionally, only from known senders)
  - ◆ outgoing, with the from address of your gateway (and optionally, only to known receivers)
- let Pluto deal with IKE
- let KLIPS deal with ESP

Since Pluto authenticates its partners during the negotiation, and KLIPS drops packets for which no tunnel has been negotiated, this may be all you need.

### A simple set of rules

In simple cases, you need only a few rules, as in this example:

```
# allow IPsec
#
# IKE negotiations
iptables -I INPUT -p udp --sport 500 --dport 500 -j ACCEPT
iptables -I OUTPUT -p udp --sport 500 --dport 500 -j ACCEPT
# ESP encryption and authentication
iptables -I INPUT -p 50 -j ACCEPT
iptables -I OUTPUT -p 50 -j ACCEPT
```

This should be all you need to allow IPsec through *lokkit*, which ships with Red Hat 9, on its medium security setting. Once you've tweaked to your satisfaction, save your active rule set with:

```
service iptables save
```

### Other rules

You can add additional rules, or modify existing ones, to work with IPsec and with your network and policies. We give a some examples in this section.

However, while it is certainly possible to create an elaborate set of rules yourself (please let us know via the [mailing list](#) if you do), it may be both easier and more secure to use a set which has already been published and tested.

The published rule sets we know of are described in the [next section](#).

#### Adding additional rules

If necessary, you can add additional rules to:

*reject IPsec packets that are not to or from known gateways*

This possibility is discussed in more detail [later](#)

*allow systems behind your gateway to build IPsec tunnels that pass through the gateway*

This possibility is discussed in more detail [later](#)

*filter incoming packets emerging from KLIPS.*

Firewall rules can recognise packets emerging from IPsec. They are marked as arriving on an interface such as *ipsec0*, rather than *eth0*, *ppp0* or whatever.

It is therefore reasonably straightforward to filter these packets in whatever way suits your situation.

#### Modifying existing rules

In some cases rules that work fine before you add IPsec may require modification to work with IPsec.

This is especially likely for rules that deal with interfaces on the Internet side of your system. IPsec adds a new interface; often the rules must change to take account of that.

## Introduction to FreeS/WAN

For example, consider the rules given in [this section](#) of the Netfilter documentation:

Most people just have a single PPP connection to the Internet, and don't want anyone coming back into their network, or the firewall:

```
## Insert connection-tracking modules (not needed if built into kernel).
# insmod ip_conntrack
# insmod ip_conntrack_ftp

## Create chain which blocks new connections, except if coming from inside.
# iptables -N block
# iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A block -m state --state NEW -i ! ppp0 -j ACCEPT
# iptables -A block -j DROP

## Jump to that chain from INPUT and FORWARD chains.
# iptables -A INPUT -j block
# iptables -A FORWARD -j block
```

On an IPsec gateway, those rules may need to be modified. The above allows new connections from *anywhere* except *ppp0*. That means new connections from *ipsec0* are allowed.

Do you want to allow anyone who can establish an IPsec connection to your gateway to initiate TCP connections to any service on your network? Almost certainly not if you are using opportunistic encryption. Quite possibly not even if you have only explicitly configured connections.

To disallow incoming connections from *ipsec0*, change the middle section above to:

```
## Create chain which blocks new connections, except if coming from inside.
# iptables -N block
# iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A block -m state --state NEW -i ppp+ -j DROP
# iptables -A block -m state --state NEW -i ipsec+ -j DROP
# iptables -A block -m state --state NEW -i -j ACCEPT
# iptables -A block -j DROP
```

The original rules accepted NEW connections from anywhere except *ppp0*. This version drops NEW connections from any PPP interface (*ppp+*) and from any ipsec interface (*ipsec+*), then accepts the survivors.

Of course, these are only examples. You will need to adapt them to your own situation.

## Published rule sets

Several sets of firewall rules that work with FreeS/WAN are available.

### Scripts based on Ranch's work

One user, Rob Hutton, posted his boot time scripts to the mailing list, and we included them in previous versions of this documentation. They are still available from our [web site](#). However, they were for an earlier FreeS/WAN version so we no longer recommend them. Also, they had some bugs. See this [message](#).

Those scripts were based on David Ranch's scripts for his "Trinity OS" for setting up a secure Linux. Check his [home page](#) for the latest version and for information on his [book](#) on securing Linux. If you are going to base your firewalling on Ranch's scripts, we recommend using his latest version, and sending him any IPsec modifications you make for incorporation into later versions.

## The Seattle firewall

We have had several mailing lists reports of good results using FreeS/WAN with Seawall (the Seattle Firewall). See that project's [home page](#) on Sourceforge.

## The RCF scripts

Another set of firewall scripts with IPsec support are the RCF or rc.firewall scripts. See their [home page](#).

## Asgard scripts

[Asgard's Realm](#) has set of firewall scripts with FreeS/WAN support, for 2.4 kernels and iptables.

## User scripts from the mailing list

One user gave considerable detail on his scripts, including supporting [IPX](#) through the tunnel. His message was too long to conveniently be quoted here, so I've put it in a [separate file](#).

# Calling firewall scripts, named in ipsec.conf(5)

The [ipsec.conf\(5\)](#) configuration file has three pairs of parameters used to specify an interface between FreeS/WAN and firewalling code.

Note that using these is not required if you have a static firewall setup. In that case, you just set your firewall up at boot time (in a way that permits the IPsec connections you want) and do not change it thereafter. Omit all the FreeS/WAN firewall parameters and FreeS/WAN will not attempt to adjust firewall rules at all. See [above](#) for some information on appropriate scripts.

However, if you want your firewall rules to change when IPsec connections change, then you need to use these parameters.

## Scripts called at IPsec start and stop

One pair of parameters are set in the *config setup* section of the [ipsec.conf\(5\)](#) file and affect all connections:

```
prepluto=  
    script to be called before pluto\(8\) IKE daemon is started.  
postpluto=  
    script to be called after pluto\(8\) IKE daemon is stopped.
```

These parameters allow you to change firewall parameters whenever IPsec is started or stopped.

They can also be used in other ways. For example, you might have *prepluto* add a module to your kernel for the secure network interface or make a dialup connection, and then have *postpluto* remove the module or take the connection down.

## Scripts called at connection up and down

The other parameters are set in connection descriptions. They can be set in individual connection descriptions, and could even call different scripts for each connection for maximum flexibility. In most applications,

however, it makes sense to use only one script and to call it from *conn %default* section so that it applies to all connections.

You can:

*either*

set *leftfirewall=yes* or *rightfirewall=yes* to use our supplied default script

*or*

assign a name in a *leftupdown=* or *rightupdown=* line to use your own script

Note that ***only one of these should be used***. You cannot sensibly use both. Since ***our default script is obsolete*** (designed for firewalls using *ipfwadm(8)* on 2.0 kernels), most users who need this service will ***need to write a custom script***.

### The default script

We supply a default script named *\_updown*.

*leftfirewall=*

*rightfirewall=*

indicates that the gateway is doing firewalling and that *pluto(8)* should poke holes in the firewall as required.

Set these to *yes* and Pluto will call our default script *\_updown* with appropriate arguments whenever it:

- starts or stops IPsec services
- brings a connection up or down

The supplied default *\_updown* script is appropriate for simple cases using the *ipfwadm(8)* firewalling package.

### User-written scripts

You can also write your own script and have Pluto call it. Just put the script's name in one of these *ipsec.conf(5)* lines:

*leftupdown=*

*rightupdown=*

specifies a script to call instead of our default script *\_updown*.

Your script should take the same arguments and use the same environment variables as *\_updown*. See the "updown command" section of the *ipsec pluto(8)* man page for details.

Note that ***you should not modify our \_updown script in place***. If you did that, then upgraded FreeS/WAN, the upgrade would install a new default script, overwriting your changes.

### Scripts for ipchains or iptables

Our *\_updown* is for firewalls using *ipfwadm(8)*, the firewall code for the 2.0 series of Linux kernels. If you are using the more recent packages *ipchains(8)* (for 2.2 kernels) or *iptables(8)* (2.4 kernels), then you must

do one of:

- use static firewall rules which are set up at boot time as described above and do not need to be changed by Pluto
- limit yourself to ipchains(8)'s ipfwadm(8) emulation mode in order to use our script
- write your own script and call it with *leftupdown* and *rightupdown*.

You can write a script to do whatever you need with firewalling. Specify its name in a *[left|right]updown=* parameter in ipsec.conf(5) and Pluto will automatically call it for you.

The arguments Pluto passes such a script are the same ones it passes to our default *\_updown* script, so the best way to build yours is to copy ours and modify the copy.

Note, however, that *you should not modify our \_updown script in place*. If you did that, then upgraded FreeS/WAN, the upgrade would install a new default script, overwriting your changes.

## A complication: IPsec vs. NAT

Network Address Translation, also known as IP masquerading, is a method of allocating IP addresses dynamically, typically in circumstances where the total number of machines which need to access the Internet exceeds the supply of IP addresses.

Any attempt to perform NAT operations on IPsec packets *between the IPsec gateways* creates a basic conflict:

- IPsec wants to authenticate packets and ensure they are unaltered on a gateway-to-gateway basis
- NAT rewrites packet headers as they go by
- IPsec authentication fails if packets are rewritten anywhere between the IPsec gateways

For AH, which authenticates parts of the packet header including source and destination IP addresses, this is fatal. If NAT changes those fields, AH authentication fails.

For IKE and ESP it is not necessarily fatal, but is certainly an unwelcome complication.

## NAT on or behind the IPsec gateway works

This problem can be avoided by having the masquerading take place *on or behind* the IPsec gateway.

This can be done physically with two machines, one physically behind the other. A picture, using SG to indicate IPsec Security Gateways, is:

```
clients --- NAT ----- SG ----- SG
                two machines
```

In this configuration, the actual client addresses need not be given in the *leftsubnet=* parameter of the FreeS/WAN connection description. The security gateway just delivers packets to the NAT box; it needs only that machine's address. What that machine does with them does not affect FreeS/WAN.

A more common setup has one machine performing both functions:

```
clients ----- NAT/SG -----SG
                one machine
```

Here you have a choice of techniques depending on whether you want to make your client subnet visible to clients on the other end:

- If you want the single gateway to behave like the two shown above, with your clients hidden behind the NAT, then omit the *leftsubnet=* parameter. It then defaults to the gateway address. Clients on the other end then talk via the tunnel only to your gateway. The gateway takes packets emerging from the tunnel, applies normal masquerading, and forwards them to clients.
- If you want to make your client machines visible, then give the client subnet addresses as the *leftsubnet=* parameter in the connection description and

*either*

set *leftfirewall=yes* to use the default *updown* script

*or*

use your own script by giving its name in a *leftupdown=* parameter

These scripts are described in their own section.

In this case, no masquerading is done. Packets to or from the client subnet are encrypted or decrypted without any change to their client subnet addresses, although of course the encapsulating packets use gateway addresses in their headers. Clients behind the right security gateway see a route via that gateway to the left subnet.

## NAT between gateways is problematic

We recommend not trying to build IPsec connections which pass through a NAT machine. This setup poses problems:

```
clients --- SG --- NAT ----- SG
```

If you must try it, some references are:

- Jean\_Francois Nadeau's document on doing IPsec behind NAT
- VPN masquerade patches to make a Linux NAT box handle IPsec packets correctly

## Other references on NAT and IPsec

Other documents which may be relevant include:

- an Internet Draft on IPsec and NAT which may eventually evolve into a standard solution for this problem.
- an informational RFC, *Security Model with Tunnel-mode IPsec for NAT Domains*.
- an article in Cisco's *Internet Protocol Journal*

## Other complications

Of course simply allowing UDP 500 and ESP packets is not the whole story. Various other issues arise in making IPsec and packet filters co-exist and even co-operate. Some of them are summarised below.

## IPsec *through* the gateway

Basic IPsec packet filtering rules deal only with packets addressed to or sent from your IPsec gateway.

## Introduction to FreeS/WAN

It is a separate policy decision whether to permit such packets to pass through the gateway so that client machines can build end-to-end IPsec tunnels of their own. This may not be practical if you are using NAT (IP masquerade) on your gateway, and may conflict with some corporate security policies.

Where possible, allowing this is almost certainly a good idea. Using IPsec on an end-to-end basis is more secure than gateway-to-gateway.

Doing it is quite simple. You just need firewall rules that allow UDP port 500 and protocols 50 and 51 to pass through your gateway. If you wish, you can of course restrict this to certain hosts.

## Preventing non-IPsec traffic

You can also filter *everything but* UDP port 500 and ESP or AH to restrict traffic to IPsec only, either for anyone communicating with your host or just for specific partners.

One application of this is for the telecommuter who might have:

```
Sunset=====West-----East ===== firewall --- the Internet
    home network      untrusted net      corporate network
```

The subnet on the right is 0.0.0.0/0, the whole Internet. The West gateway is set up so that it allows only IPsec packets to East in or out.

This configuration is used in AT&T Research's network. For details, see the [papers](#) links in our introduction.

Another application would be to set up firewall rules so that an internal machine, such as an employees-only web server, could not talk to the outside world except via specific IPsec tunnels.

## Filtering packets from unknown gateways

It is possible to use firewall rules to restrict UDP 500, ESP and AH packets so that these packets are accepted only from known gateways. This is not strictly necessary since FreeS/WAN will discard packets from unknown gateways. You might, however, want to do it for any of a number of reasons. For example:

- Arguably, "belt and suspenders" is the sensible approach to security. If you can block a potential attack in two ways, use both. The only question is whether to look for a third way after implementing the first two.
- Some admins may prefer to use the firewall code this way because they prefer firewall logging to FreeS/WAN's logging.
- You may need it to implement your security policy. Consider an employee working at home, and a policy that says traffic from the home system to the Internet at large must go first via IPsec to the corporate LAN and then out to the Internet via the corporate firewall. One way to do that is to make *ipsec0* the default route on the home gateway and provide exceptions only for UDP 500 and ESP to the corporate gateway. Everything else is then routed via the tunnel to the corporate gateway.

It is not possible to use only static firewall rules for this filtering if you do not know the other gateways' IP addresses in advance, for example if you have "road warriors" who may connect from a different address each time or if want to do opportunistic encryption to arbitrary gateways. In these cases, you can accept UDP 500 IKE packets from anywhere, then use the updown script feature of pluto(8) to dynamically adjust firewalling for each negotiated tunnel.

Firewall packet filtering does not much reduce the risk of a denial of service attack on FreeS/WAN. The firewall can drop packets from unknown gateways, but KLIPS does that quite efficiently anyway, so you gain little. The firewall cannot drop otherwise legitimate packets that fail KLIPS authentication, so it cannot protect against an attack designed to exhaust resources by making FreeS/WAN perform many expensive authentication operations.

In summary, firewall filtering of IPsec packets from unknown gateways is possible but not strictly necessary.

## Other packet filters

When the IPsec gateway is also acting as your firewall, other packet filtering rules will be in play. In general, those are outside the scope of this document. See our [Linux firewall links](#) for information. There are a few types of packet, however, which can affect the operation of FreeS/WAN or of diagnostic tools commonly used with it. These are discussed below.

### ICMP filtering

ICMP is the *Internet Control Message Protocol*. It is used for messages between IP implementations themselves, whereas IP used is used between the clients of those implementations. ICMP is, unsurprisingly, used for control messages. For example, it is used to notify a sender that a destination is not reachable, or to tell a router to reroute certain packets elsewhere.

ICMP handling is tricky for firewalls.

- You definitely want some ICMP messages to get through; things won't work without them. For example, your clients need to know if some destination they ask for is unreachable.
- On the other hand, you do equally definitely do not want untrusted folk sending arbitrary control messages to your machines. Imagine what someone moderately clever and moderately malicious could do to you, given control of your network's routing.

ICMP does not use ports. Messages are distinguished by a "message type" field and, for some types, by an additional "code" field. The definitive list of types and codes is on the [IANA](#) site.

One expert uses this definition for ICMP message types to be dropped at the firewall.

```
# ICMP types which lack socially redeeming value.
# 5      Redirect
# 9      Router Advertisement
# 10     Router Selection
# 15     Information Request
# 16     Information Reply
# 17     Address Mask Request
# 18     Address Mask Reply

badicmp='5 9 10 15 16 17 18'
```

A more conservative approach would be to make a list of allowed types and drop everything else.

Whichever way you do it, your ICMP filtering rules on a FreeS/WAN gateway should allow at least the following ICMP packet types:

*echo (type 8)*

### *echo reply (type 0)*

These are used by ping(1). We recommend allowing both types through the tunnel and to or from your gateway's external interface, since ping(1) is an essential testing tool.

It is fairly common for firewalls to drop ICMP echo packets addressed to machines behind the firewall. If that is your policy, please create an exception for such packets arriving via an IPsec tunnel, at least during initial testing of those tunnels.

### *destination unreachable (type 3)*

This is used, with code 4 (Fragmentation Needed and Don't Fragment was Set) in the code field, to control path MTU discovery. Since IPsec processing adds headers, enlarges packets and may cause fragmentation, an IPsec gateway should be able to send and receive these ICMP messages *on both inside and outside interfaces*.

## UDP packets for traceroute

The traceroute(1) utility uses UDP port numbers from 33434 to approximately 33633. Generally, these should be allowed through for troubleshooting.

Some firewalls drop these packets to prevent outsiders exploring the protected network with traceroute(1). If that is your policy, consider creating an exception for such packets arriving via an IPsec tunnel, at least during initial testing of those tunnels.

## UDP for L2TP

Windows 2000 does, and products designed for compatibility with it may, build L2TP tunnels over IPsec connections.

For this to work, you must allow UDP protocol 1701 packets coming out of your tunnels to continue to their destination. You can, and probably should, block such packets to or from your external interfaces, but allow them from *ipsec0*.

See also our Windows 2000 interoperation discussion.

## How it all works: IPsec packet details

IPsec uses three main types of packet:

### *IKE* uses *the UDP protocol and port 500*.

Unless you are using only (less secure, not recommended) manual keying, you need IKE to negotiate connection parameters, acceptable algorithms, key sizes and key setup. IKE handles everything required to set up, rekey, repair or tear down IPsec connections.

### *ESP* is *protocol number 50*

This is required for encrypted connections.

### *AH* is *protocol number 51*

This can be used where only authentication, not encryption, is required.

All of those packets should have appropriate IPsec gateway addresses in both the to and from IP header fields. Firewall rules can check this if you wish, though it is not strictly necessary. This is discussed in more detail later.

IPsec processing of incoming packets authenticates them then removes the ESP or AH header and decrypts if necessary. Successful processing exposes an inner packet which is then delivered back to the firewall machinery, marked as having arrived on an *ipsec[0-3]* interface. Firewall rules can use that interface label to distinguish these packets from unencrypted packets which are labelled with the physical interface they arrived on (or perhaps with a non-IPsec virtual interface such as *ppp0*).

One of our users sent a mailing list message with a diagram of the packet flow.

## ESP and AH do not have ports

Some protocols, such as TCP and UDP, have the notion of ports. Others protocols, including ESP and AH, do not. Quite a few IPsec newcomers have become confused on this point. There are no ports *in* the ESP or AH protocols, and no ports used *for* them. For these protocols, *the idea of ports is completely irrelevant*.

## Header layout

The protocol numbers for ESP or AH are used in the 'next header' field of the IP header. On most non-IPsec packets, that field would have one of:

- 1 for ICMP
- 4 for IP-in-IP encapsulation
- 6 for TCP
- 17 for UDP
- ... or one of about 100 other possibilities listed by IANA

Each header in the sequence tells what the next header will be. IPsec adds headers for ESP or AH near the beginning of the sequence. The original headers are kept and the 'next header' fields adjusted so that all headers can be correctly interpreted.

For example, using *[ ]* to indicate data protected by ESP and unintelligible to an eavesdropper between the gateways:

- a simple packet might have only IP and TCP headers with:
  - ◆ IP header says next header --> TCP
  - ◆ TCP header port number --> which process to send data to
  - ◆ data
- with ESP transport mode encapsulation, that packet would have:
  - ◆ IP header says next header --> ESP
  - ◆ ESP header *[* says next --> TCP
  - ◆ TCP header port number --> which process to send data to
  - ◆ data *]*

Note that the IP header is outside ESP protection, visible to an attacker, and that the final destination must be the gateway.

- with ESP in tunnel mode, we might have:
  - ◆ IP header says next header --> ESP
  - ◆ ESP header *[* says next --> IP
  - ◆ IP header says next header --> TCP
  - ◆ TCP header port number --> which process to send data to
  - ◆ data *]*

## Introduction to FreeS/WAN

Here the inner IP header is protected by ESP, unreadable by an attacker. Also, the inner header can have a different IP address than the outer IP header, so the decrypted packet can be routed from the IPsec gateway to a final destination which may be another machine.

Part of the ESP header itself is encrypted, which is why the `[` indicating protected data appears in the middle of some lines above. The next header field of the ESP header is protected. This makes traffic analysis more difficult. The next header field would tell an eavesdropper whether your packet was UDP to the gateway, TCP to the gateway, or encapsulated IP. It is better not to give this information away. A clever attacker may deduce some of it from the pattern of packet sizes and timings, but we need not make it easy.

IPsec allows various combinations of these to match local policies, including combinations that use both AH and ESP headers or that nest multiple copies of these headers.

For example, suppose my employer has an IPsec VPN running between two offices so all packets travelling between the gateways for those offices are encrypted. If gateway policies allow it (The admins could block UDP 500 and protocols 50 and 51 to disallow it), I can build an IPsec tunnel from my desktop to a machine in some remote office. Those packets will have one ESP header throughout their life, for my end-to-end tunnel. For part of the route, however, they will also have another ESP layer for the corporate VPN's encapsulation. The whole header scheme for a packet on the Internet might be:

- IP header (with gateway address) says next header --> ESP
- ESP header `[` says next --> IP
- IP header (with receiving machine address) says next header --> ESP
- ESP header `[` says next --> TCP
- TCP header port number --> which process to send data to
- data `]]`

The first ESP (outermost) header is for the corporate VPN. The inner ESP header is for the secure machine-to-machine link.

## DHR on the updown script

Here are some mailing list comments from pluto(8) developer Hugh Redelmeier on an earlier draft of this document:

There are many important things left out

- firewalling is important but must reflect (implement) policy. Since policy isn't the same for all our customers, and we're not experts, we should concentrate on FW and MASQ interactions with FreeS/WAN.
- we need a diagram to show packet flow WITHIN ONE MACHINE, assuming IKE, IPsec, FW, and MASQ are all done on that machine. The flow is obvious if the components are run on different machines (trace the cables).

IKE input:

- + packet appears on public IF, as UDP port 500
- + input firewalling rules are applied (may discard)
- + Pluto sees the packet.

IKE output:

- + Pluto generates the packet & writes to public IF, UDP port 500
- + output firewalling rules are applied (may discard)

## Introduction to FreeS/WAN

- + packet sent out public IF

IPsec input, with encapsulated packet, outer destination of this host:

- + packet appears on public IF, protocol 50 or 51. If this packet is the result of decapsulation, it will appear instead on the paired ipsec IF.
- + input firewalling rules are applied (but packet is opaque)
- + KLIPS decapsulates it, writes result to paired ipsec IF
- + input firewalling rules are applied to resulting packet as input on ipsec IF
- + if the destination of the packet is this machine, the packet is passed on to the appropriate protocol handler. If the original packet was encapsulated more than once and the new outer destination is this machine, that handler will be KLIPS.
- + otherwise:
  - \* routing is done for the resulting packet. This may well direct it into KLIPS for encoding or encrypting. What happens then is described elsewhere.
  - \* forwarding firewalling rules are applied
  - \* output firewalling rules are applied
  - \* the packet is sent where routing specified

IPsec input, with encapsulated packet, outer destination of another host:

- + packet appears on some IF, protocol 50 or 51
- + input firewalling rules are applied (but packet is opaque)
- + routing selects where to send the packet
- + forwarding firewalling rules are applied (but packet is opaque)
- + packet forwarded, still encapsulated

IPsec output, from this host or from a client:

- + if from a client, input firewalling rules are applied as the packet arrives on the private IF
- + routing directs the packet to an ipsec IF (this is how the system decides KLIPS processing is required)
- + if from a client, forwarding firewalling rules are applied
- + KLIPS eroute mechanism matches the source and destination to registered eroutes, yielding a SPI group. This dictates processing, and where the resulting packet is to be sent (the destinations SG and the nexthop).
- + output firewalling is not applied to the resulting encapsulated packet

- Until quite recently, KLIPS would double encapsulate packets that didn't strictly need to be. Firewalling should be prepared for those packets showing up as ESP and AH protocol input packets on an ipsec IF.
- MASQ processing seems to be done as if it were part of the forwarding firewall processing (this should be verified).
- If a firewall is being used, it is likely the case that it needs to be adjusted whenever IPsec SAs are added or removed. Pluto invokes a script to do this (and to adjust routing) at suitable times. The default script is only suitable for ipfwadm-managed firewalls. Under LINUX 2.2.x kernels, ipchains can be managed by ipfwadm (emulation), but ipchains more powerful if manipulated using the ipchains command. In this case, a custom updown script must be used.

We think that the flexibility of ipchains precludes us supplying an updown script that would be widely appropriate.

# Linux FreeS/WAN Troubleshooting Guide

## Overview

This document covers several general places where you might have a problem:

1. During install.
2. During the negotiation process.
3. Using an established connection.

This document also contains notes which expand on points made in these sections, and tips for problem reporting. If the other end of your connection is not FreeS/WAN, you'll also want to read our interoperation document.

## 1. During Install

### 1.1 RPM install gotchas

With the RPM method:

- Be sure you have installed both the userland tools and the kernel components. One will not work without the other. For example, when using FreeS/WAN-produced RPMs for our 2.03 release, you need both:

```
freeswan-userland-2.03_2.4.20_20.9-0.i386.rpm  
freeswan-module-2.03_2.4.20_20.9-0.i386.rpm
```

### 1.2 Problems installing from source

When installing from source, you may find these problems:

- Missing library. See this FAQ.
- Missing utilities required for compile. See this checklist.
- Kernel version incompatibility. See this FAQ.
- Another compile problem. Find information in the out.\* files, ie. out.kpatch, out.kbuild, created at compile time in the top-level Linux FreeS/WAN directory. Error messages generated by KLIPS during the boot sequence are accessible with the *dmesg* command. Check the list archives and the List in Brief to see if this is a known issue. If it is not, report it to the bugs list as described in our problem reporting section. In some cases, you may be asked to provide debugging information using *gdb*; details below.
- If your kernel compiles but you fail to install your new FreeS/WAN-enabled kernel, review the sections on installing the patched kernel, and testing to see if install succeeded.

### 1.3 Install checks

*ipsec verify* checks a number of FreeS/WAN essentials. Here are some hints on what to do when your system doesn't check out:

<i><b>Problem</b></i>	<i><b>Status</b></i>	<i><b>Action</b></i>
<i>ipsec</i> not on-path		Add <code>/usr/local/sbin</code> to your PATH.
Missing KLIPS support	critical	See <a href="#">this FAQ</a> .
No RSA private key		Follow <a href="#">these instructions</a> to create an RSA key pair for your host. RSA keys are: <ul style="list-style-type: none"> <li>• required for opportunistic encryption, and</li> <li>• our preferred method to authenticate pre-configured connections.</li> </ul>
<i>pluto</i> not running	critical	<code>service ipsec start</code>
No port 500 hole	critical	Open port 500 for IKE negotiation.
Port 500 check N/A		Check that port 500 is open for IKE negotiation.
Failed DNS checks		Opportunistic encryption requires information from DNS. To set this up, see <a href="#">our instructions</a> .
No public IP address		Check that the interface which you want to protect with IPsec is up and running.

### 1.3 Troubleshooting OE

OE should work with no local configuration, if you have posted DNS TXT records according to the instructions in our [quickstart guide](#). If you encounter trouble, try these hints. We welcome additional hints via the [users' mailing list](#).

<i><b>Symptom</b></i>	<i><b>Problem</b></i>	<i><b>Action</b></i>
<p>You're running FreeS/WAN 2.01 (or later), and initiating a connection to FreeS/WAN 2.00 (or earlier). In your logs, you see a message like:</p> <pre>no RSA public key known for '192.0.2.13'; DNS search for KEY failed (no KEY record for 13.2.0.192.in-addr.arpa.)</pre> <p>The older FreeS/WAN logs no error.</p>	<p>A protocol level incompatibility between 2.01 (or later) and 2.00 (or earlier) causes this error. It occurs when a FreeS/WAN 2.01 (or later) box for which no KEY record is posted attempts to initiate an OE connection to older FreeS/WAN versions (2.00 and earlier). Note that older versions can initiate to</p>	<p>If you control the peer host, upgrade its FreeS/WAN to 2.01 (or later), and post new style TXT records for it. If not, but if you know its sysadmin, perhaps a quick note is in order. If neither option is possible, you can ease the transition by posting an old style KEY record (created with a command like <code>"ipsec showhostkey --key"</code>) to the reverse map for the FreeS/WAN 2.01 (or later) box.</p>

## Introduction to FreeS/WAN

	newer versions without this error.	
OE host is very slow to contact other hosts.	Slow DNS service while running OE.	It's a good idea to run a caching DNS server on your OE host, as outlined in <a href="#">this mailing list message</a> . If your DNS servers are elsewhere, put their IPs in the <i>clear</i> policy group, and re-read groups with  <code>ipsec auto --rereadgroups</code>
Can't Opportunistically initiate for 192.0.2.2 to 192.0.2.3: no TXT record for 13.2.0.192.in-addr.arpa.	Peer is not set up for OE.	None. Plenty of hosts on the Internet do not run OE. If, however, you have set OE up on that peer, this may indicate that you need to wait up to 48 hours for its DNS records to propagate.
<i>ipsec verify</i> does not find DNS records:  ... Looking for TXT in forward map: xy.example.com...[FAILED] Looking for TXT in reverse map...[FAILED] ...  You also experience authentication failure:  Possible authentication failure: no acceptable response to our first encrypted message	DNS records are not posted or have not propagated.	Did you post the DNS records necessary for OE? If not, do so using the instructions in our <a href="#">quickstart guide</a> . If so, wait up to 48 hours for the DNS records to propagate.
<i>ipsec verify</i> does not find DNS records, and you experience authentication failure.	For iOE, your ID does not match location of forward DNS record.	In <i>config setup</i> , change <i>myid=</i> to match the forward DNS where you posted the record. Restart FreeS/WAN. For reference, see our <a href="#">iOE instructions</a> .
<i>ipsec verify</i> finds DNS records, yet there is still authentication failure. ( ? )	DNS records are malformed.	Re-create the records and send new copies to your DNS administrator.
<i>ipsec verify</i> finds DNS records, yet there is still authentication failure. ( ? )	DNS records show different keys for a gateway vs. its subnet hosts.	All TXT records for boxes protected by an OE gateway must contain the gateway's public key. Re-create and re-post any incorrect records using <a href="#">these instructions</a> .
OE gateway loses connectivity to its subnet. The gateway's routing table shows routes to the subnet through IPsec interfaces.	The subnet is part of the <i>private</i> or <i>block</i> policy group on the gateway.	Remove the subnet from the group, and reread groups with  <code>ipsec auto --rereadgroups</code>
OE does not work to hosts on the local LAN.	This is a known issue.	See <a href="#">this list</a> of known issues with OE.
FreeS/WAN does not seem to be executing your default policy. In your logs, you see a message like:	<b>Fullnet</b> in a policy group file defines	Find all policies which contain fullnet with:  <code>grep -F 0.0.0.0/0 /etc/ipsec.d/policies/*</code>

<pre>/etc/ipsec.d/policies/ipprivate-or-clear" line 14: subnet "0.0.0.0/0", source 192.0.2.13/32, already "private-or-clear"</pre>	<p>your default policy. Fullnet should normally be present in only one policy group file. The fine print: you can have two default policies defined so long as they protect different local endpoints (e.g. the FreeS/WAN gateway and a subnet).</p>	<p>then remove the unwanted occurrence(s).</p>
--	--	--

## 2. During Negotiation

When you fail to bring up a tunnel, you'll need to find out:

- what your connection state is, and often
- an error message.

before you can diagnose your problem.

### 2.1 Determine Connection State

#### Finding current state

You can see connection states (STATE\_MAIN\_I1 and so on) when you bring up a connection on the command line. If you have missed this, or brought up your connection automatically, use:

```
ipsec auto --status
```

The most relevant state is the last one reached.

#### ***What's this supposed to look like?***

Negotiations should proceed through various states, in the processes of:

1. IKE negotiations (aka Phase 1, Main Mode, STATE\_MAIN\_\*)
2. IPSEC negotiations (aka Phase 2, Quick Mode, STATE\_QUICK\_\*)

These are done and a connection is established when you see messages like:

```
000 #21: "myconn" STATE_MAIN_I4 (ISAKMP SA established)...
000 #2: "myconn" STATE_QUICK_I2 (sent QI2, IPsec SA established)...
```

Look for the key phrases are "ISAKMP SA established" and "IPSec SA established", with the relevant connection name. Often, this happens at STATE\_MAIN\_I1 and STATE\_QUICK\_I2, respectively.

*ipsec auto --status* will tell you what states *have been achieved*, rather than the current state. Since determining the current state is rather more difficult to do, current state information is not available from Linux FreeS/WAN. If you are actively bringing a connection up, the status report's last states for that connection likely reflect its current state. Beware, though, of the case where a connection was correctly brought up but is now downed: Linux FreeS/WAN will not notice this until it attempts to rekey. Meanwhile, the last known state indicates that the connection has been established.

If your connection is stuck at STATE\_MAIN\_I1, skip straight to [here](#).

## 2.2 Finding error text

Solving most errors will require you to find verbose error text, either on the command line or in the logs.

### Verbose start for more information

Note that you can get more detail from *ipsec auto* using the `--verbose` flag:

```
ipsec auto --verbose --up west-east
```

More complete information can be gleaned from the [log files](#).

### Debug levels count

The amount of description you'll get here depends on `ipsec.conf` debug settings, *klipsdebug*= and *plutodebug*=. When troubleshooting, set at least one of these to *all*, and when done, reset it to *none* so your logs don't fill up. Note that you must have enabled the [klipsdebug compile-time option](#) for the *klipsdebug* configuration switch to work.

For negotiation problems *plutodebug* is most relevant. *klipsdebug* applies mainly to attempts to use an already-established connection. See also [this](#) description of the division of duties within Linux FreeS/WAN.

After raising your debug levels, restart Linux FreeS/WAN to ensure that `ipsec.conf` is reread, then recreate the error to generate verbose logs.

### *ipsec barf* for lots of debugging information

*ipsec barf (8)* collects a bunch of useful debugging information, including these logs Use the command

```
ipsec barf > barf.west
```

to generate one.

### Find the error

Search out the failure point in your logs. Are there a handful of lines which succinctly describe how things are going wrong or contrary to your expectation? Sometimes the failure point is not immediately obvious: Linux FreeS/WAN's errors are usually not marked "Error". Have a look in the [FAQ](#) for what some common failures look like.

Tip: problems snowball. Focus your efforts on the first problem, which is likely to be the cause of later errors.

### Play both sides

Also find error text on the peer IPsec box. This gives you two perspectives on the same failure.

At times you will require information which only one side has. The peer can merely indicate the presence of an error, and its approximate point in the negotiations. If one side keeps retrying, it may be because there is a show stopper on the other side. Have a look at the other side and figure out what it doesn't like.

If the other end is not Linux FreeS/WAN, the principle is the same: replicate the error with its most verbose logging on, and capture the output to a file.

## 2.3 Interpreting a Negotiation Error

### Connection stuck at STATE\_MAIN\_I1

This error commonly happens because IKE (port 500) packets, needed to negotiate an IPsec connection, cannot travel freely between your IPsec gateways. See [our firewall document](#) for details.

### Other errors

Other errors require a bit more digging. Use the following resources:

- [the FAQ](#) . Since this document is constantly updated, the snapshot's FAQ may have a new entry relevant to your problem.
- [our background document](#) . Special considerations which, while not central to Linux FreeS/WAN, are often tripped over. Includes problems with [packet fragmentation](#), and considerations for testing opportunism.
- [the list archives](#). Each of the searchable archives works differently, so it's worth checking each. Use a search term which is generic, but identifies your error, for example "No connection is known for". Often, you will find that your question has been answered in the past. Finding an archived answer is quicker than asking the list. You may, however, find similar questions without answers. If you do, send their URLs to the list with your trouble report. The additional examples may help the list tech support person find your answer.
- Look into the code where the error is being generated. The pluto code is nicely documented with comments and meaningful variable names.

If you have failed to solve your problem with the help of these resources, send a detailed problem report to the users list, following these [guidelines](#).

## 3. Using a Connection

### 3.1 Orienting yourself

#### *How do I know if it works?*

Test your connection by sending packets through it. The simplest way to do this is with ping, but the ping needs to *test the correct tunnel*. See [this example scenario](#) if you don't understand this.

If your ping returns, test any other connections you've brought u all check out, great. You may wish to test with large packets for MTU problems.

### ***ipsec barf* is useful again**

If your ping fails to return, generate an ipsec barf debugging report on each IPSec gateway. On a non-Linux FreeS/WAN implementation, gather equivalent information. Use this, and the tips in the next sections, to troubleshoot. Are you sure that both endpoints are capable of hearing and responding to ping?

## **3.2 Those pesky configuration errors**

IPSec may be dropping your ping packets since they do not belong in the tunnels you have constructed:

- Your ping may not test the tunnel you intend to test. For details, see our "I can't ping" FAQ.
- Alternately, you may have a configuration error. For example, you may have configured one of the four possible tunnels between two gateways, but not the one required to secure the important traffic you're now testing. In this case, add and start the tunnel, and try again.

In either case, you will often see a message like:

```
klipsdebug... no eroute
```

which we discuss in this FAQ.

Note:

- Network Address Translation (NAT) and IP masquerade may have an effect on which tunnels you need to configure.
- When testing a tunnel that protects a multi-node subnet, try several subnet nodes as ping targets, in case one node is routing incorrectly.

## **3.3 Check Routing and Firewalling**

If you've confirmed your configuration assumptions, the problem is almost certainly with routing or firewalling. Isolate the problem using interface statistics, firewall statistics, or a packet sniffer.

### **Background:**

- Linux FreeS/WAN supplies all the special routing it needs; you need only route packets out through your IPSec gateway. Verify that on the *subnetted* machines you are using for your ping-test, your routing is as expected. I have seen a tunnel "fail" because the subnet machine sending packets out an alternate gateway (not our IPSec gateway) on their return path.
- Linux FreeS/WAN requires particular firewalling considerations. Check the firewall rules on your IPSec gateways and ensure that they allow IPSec traffic through. Be sure that no other machine – for example a router between the gateways – is blocking your IPSec packets.

### **View Interface and Firewall Statistics**

Interface reports and firewall statistics can help you track down lost packets at a glance. Check any firewall statistics you may be keeping on your IPSec gateways, for dropped packets.

**Tip:** You can take a snapshot of the packets processed by your firewall with:

```
iptables -L -n -v
```

You can get creative with "diff" to find out what happens to a particular packet during transmission.

Both *cat /proc/net/dev* and *ifconfig* display interface statistics, and both are included in *ipsec barf*. Use either to check if any interface has dropped packets. If you find that one has, test whether this is related to your ping. While you ping continuously, print that interface's statistics several times. Does its drop count increase in proportion to the ping? If so, check why the packets are dropped there.

To do this, look at the firewall rules that apply to that interface. If the interface is an IPSec interface, more information may be available in the log. Grep for the word "drop" in a log which was created with *klipsdebug=all* as the error happened.

See also this [discussion](#) on interpreting *ifconfig* statistics.

### 3.4 When in doubt, sniff it out

If you have checked configuration assumptions, routing, and firewall rules, and your interface statistics yield no clue, it remains for you to investigate the mystery of the lost packet by the most thorough method: with a packet sniffer (providing, of course, that this is legal where you are working).

In order to detect packets on the ipsec virtual interfaces, you will need an up-to-date sniffer (tcpdump, ethereal, ksnuffle) on your IPSec gateway machines. You may also find it useful to sniff the ping endpoints.

#### Anticipate your packets' path

Ping, and examine each interface along the projected path, checking for your ping's arrival. If it doesn't get to the the next stop, you have narrowed down where to look for it. In this way, you can isolate a problem area, and narrow your troubleshooting focus.

Within a machine running Linux FreeS/WAN, this [packet flow diagram](#) will help you anticipate a packet's path.

Note that:

- from the perspective of the tunneled packet, the entire tunnel is one hop. That's explained in [this](#) FAQ.
- an encapsulated IPSec packet will look different, when sniffed, from the plaintext packet which generated it. You can see plaintext packets entering an IPSec interface and the resulting cyphertext packets as they emerge from the corresponding physical interface.

Once you isolate where the packet is lost, take a closer look at firewall rules, routing and configuration assumptions as they affect that specific area. If the packet is lost on an IPSec gateway, comb through *klipsdebug* output for anomalies.

If the packet goes through both gateways successfully and reaches the ping target, but does not return, suspect routing. Check that the ping target routes packets back to the IPSec gateway.

## 3.5 Check your logs

Here, too, log information can be useful. Start with the [guidelines above](#).

For connection use problems, set *klipsdebug=all*. Note that you must have enabled the *klipsdebug compile-time option* to do this. Restart Linux FreeS/WAN so that it rereads *ipsec.conf*, then recreate the error condition. When searching through *klipsdebug* data, look especially for the keywords "drop" (as in dropped packets) and "error".

Often the problem with connection use is not software error, but rather that the software is behaving contrary to expectation.

### Interpreting log text

To interpret the Linux FreeS/WAN log text you've found, use the same resources as indicated for troubleshooting connection negotiation: [the FAQ](#), our [background document](#), and the [list archives](#). Looking in the KLIPS code is only for the very brave.

If you are still stuck, send a [detailed problem report](#) to the users' list.

## 3.6 More testing for the truly thorough

### Large Packets

If each of your connections passed the ping test, you may wish to test by pinging with large packets (2000 bytes or larger). If it does not return, suspect MTU issues, and see this [discussion](#).

### Stress Tests

In most users' view, a simple ping test, and perhaps a large-packet ping test suffice to indicate a working IPSec connection.

Some people might like to do additional stress tests prior to production use. They may be interested in this [testing protocol](#) we use at interoperation conferences, aka "bakeoffs". We also have a *testing* directory that ships with the release.

## 4. Problem Reporting

### 4.1 How to ask for help

Ask for troubleshooting help on the users' mailing list, [users@lists.freeswan.org](mailto:users@lists.freeswan.org). While sometimes an initial query with a quick description of your intent and error will twig someone's memory of a similar problem, it's often necessary to send a second mail with a complete problem report.

When reporting problems to the mailing list(s), please include:

- a brief description of the problem
- if it's a compile problem, the actual output from make, showing the problem. Try to edit it down to only the relevant part, but when in doubt, be as complete as you can. If it's a kernel compile problem,

any relevant out.\* files

- if it's a run-time problem, pointers to where we can find the complete output from "ipsec barf" from BOTH ENDS (not just one of them). Remember that it's common outside the US and Canada to pay for download volume, so if you can't post barfs on the web and send the URL to the mailing list, at least compress them with tar or gzip.  
If you can, try to simplify the case that is causing the problem. In particular, if you clear your logs, start FreeS/WAN with no other connections running, cause the problem to happen, and then do *ipsec barf* on both ends immediately, that gives the smallest and least cluttered output.
- any other error messages, complaints, etc. that you saw. Please send the complete text of the messages, not just a summary.
- what your network setup is. Include subnets, gateway addresses, etc. A schematic diagram is a good format for this information.
- exactly what you were trying to do with Linux FreeS/WAN, and exactly what went wrong
- a fix, if you have one. But remember, you are sending mail to people all over the world; US residents and US citizens in particular, please read [doc/exportlaws.html](http://doc/exportlaws.html) before sending code — even small bug fixes — to the list or to us.
- When in doubt about whether to include some seemingly-trivial item of information, include it. It is rare for problem reports to have too much information, and common for them to have too little.

Here are some good general guidelines on bug reporting: [How To Ask Questions The Smart Way](#) and [How to Report Bugs Effectively](#).

## 4.2 Where to ask

To report a problem, send mail about it to the users' list. If you are certain that you have found a bug, report it to the bugs list. If you encounter a problem while doing your own coding on the Linux FreeS/WAN codebase and think it is of interest to the design team, notify the design list. When in doubt, default to the users' list. More information about the mailing lists is found [here](#).

For a number of reasons — including export-control regulations affecting almost any *private* discussion of encryption software — we prefer that problem reports and discussions go to the lists, not directly to the team. Beware that the list goes worldwide; US citizens, read this important information about your [export laws](#). If you're using this software, you really should be on the lists. To get onto them, visit [lists.freeswan.org](http://lists.freeswan.org).

If you do send private mail to our coders or want a private reply from them, please make sure that the return address on your mail (From or Reply-To header) is a valid one. They have more important things to do than to unravel addresses that have been mangled in an attempt to confuse spammers.

## 5. Additional Notes on Troubleshooting

The following sections supplement the Guide: [information available on your system](#); [testing between security gateways](#); [ifconfig reports for KLIPS debugging](#); [using GDB on Pluto](#).

### 5.1 Information available on your system

#### Logs used

Linux FreeS/WAN logs to:

- /var/log/secure (or, on Debian, /var/log/auth.log)

- /var/log/messages

Check both places to get full information. If you find nothing, check your *syslogd.conf(5)* to see where your /etc/syslog.conf or equivalent is directing *authpriv* messages.

### man pages provided

#### *ipsec.conf(5)*

Manual page for IPSEC configuration file.

#### *ipsec(8)*

Primary man page for ipsec utilities.

Other man pages are on [this list](#) and in

- /usr/local/man/man3
- /usr/local/man/man5
- /usr/local/man/man8/ipsec\_\*

### Status information

#### *ipsec auto --status*

Command to get status report from running system. Displays Pluto's state. Includes the list of connections which are currently "added" to Pluto's internal database; lists state objects reflecting ISAKMP and IPsec SAs being negotiated or installed.

#### *ipsec look*

Brief status info.

#### *ipsec barf*

Copious debugging info.

## 5.2 Testing between security gateways

Sometimes you need to test a subnet-subnet tunnel. This is a tunnel between two security gateways, which protects traffic on behalf of the subnets behind these gateways. On this network:

```
Sunset=====West-----East=====Sunrise
                IPsec gateway      IPsec gateway
                local net      untrusted net      local net
```

you might name this tunnel sunset-sunrise. You can test this tunnel by having a machine behind one gateway ping a machine behind the other gateway, but this is not always convenient or even possible.

Simply pinging one gateway from the other is not useful. Such a ping does not normally go through the tunnel. ***The tunnel handles traffic between the two protected subnets, not between the gateways***. Depending on the routing in place, a ping might

- either succeed by finding an unencrypted route
- or fail by finding no route. Packets without an IPSEC eroute are discarded.

***Neither event tells you anything about the tunnel.*** You can explicitly create an eroute to force such packets through the tunnel, or you can create additional tunnels as described in our [configuration document](#), but those may be unnecessary complications in your situation.

The trick is to explicitly test between *both gateways' private-side IP addresses*. Since the private-side interfaces are on the protected subnets, the resulting packets do go via the tunnel. Use either `ping -I` or `traceroute -i`, both of which allow you to specify a source interface. (Note: unsupported on older Linuxes). The same principles apply for a road warrior (or other) case where only one end of your tunnel is a subnet.

## 5.3 ifconfig reports for KLIPS debugging

When diagnosing problems using ifconfig statistics, you may wonder what type of activity increments a particular counter for an ipsecN device. Here's an index, posted by KLIPS developer Richard Guy Briggs:

Here is a catalogue of the types of errors that can occur for which statistics are kept when transmitting and receiving packets via klips. I notice that they are not necessarily logged in the right counter.

. . .

Sources of ifconfig statistics for ipsec devices

rx-errors:

- packet handed to ipsec\_rcv that is not an ipsec packet.
- ipsec packet with payload length not modulo 4.
- ipsec packet with bad authenticator length.
- incoming packet with no SA.
- replayed packet.
- incoming authentication failed.
- got esp packet with length not modulo 8.

tx\_dropped:

- cannot process ip\_options.
- packet ttl expired.
- packet with no route.
- route with no SA.
- cannot allocate sk\_buff.
- cannot allocate kernel memory.
- sk\_buff internal error.

The standard counters are:

```
struct enet_statistics
{
    int      rx_packets;           /* total packets received */
    int      tx_packets;           /* total packets transmitted */
    int      rx_errors;            /* bad packets received */
    int      tx_errors;            /* packet transmit problems */
    int      rx_dropped;           /* no space in linux buffers */
    int      tx_dropped;           /* no space available in linux */
    int      multicast;            /* multicast packets received */
    int      collisions;

    /* detailed rx_errors: */
    int      rx_length_errors;
    int      rx_over_errors;        /* receiver ring buff overflow */
    int      rx_crc_errors;         /* recv'd pkt with crc error */
    int      rx_frame_errors;       /* recv'd frame alignment error */
    int      rx_fifo_errors;        /* recv'r fifo overrun */
    int      rx_missed_errors;      /* receiver missed packet */

    /* detailed tx_errors */
    int      tx_aborted_errors;
    int      tx_carrier_errors;
```

```
int      tx_fifo_errors;  
int      tx_heartbeat_errors;  
int      tx_window_errors;  
};
```

of which I think only the first 6 are useful.

### 5.4 Using GDB on Pluto

You may need to use the GNU debugger, `gdb(1)`, on Pluto. This should be necessary only in unusual cases, for example if you encounter a problem which the Pluto developer cannot readily reproduce or if you are modifying Pluto.

Here are the Pluto developer's suggestions for doing this:

Can you get a core dump and use `gdb` to find out what Pluto was doing when it died?

To get a core dump, you will have to set `dumpdir` to point to a suitable directory (see [`ipsec.conf\(5\)`](#)).

To get `gdb` to tell you interesting stuff:

```
$ script  
$ cd dump-directory-you-chose  
$ gdb /usr/local/lib/ipsec/pluto core  
(gdb) where  
(gdb) quit  
$ exit
```

The resulting output will have been captured by the `script` command in a file called "typescript". Send it to the list.

Do not delete the core file. I may need to ask you to print out some more relevant stuff.

Note that the *dumpdir* parameter takes effect only when the IPsec subsystem is restarted — reboot or `ipsec setup restart`.

# Linux FreeS/WAN Compatibility Guide

Much of this document is quoted directly from the Linux FreeS/WAN [mailing list](#). Thanks very much to the community of testers, patchers and commenters there, especially the ones quoted below but also various contributors we haven't quoted.

## Implemented parts of the IPsec Specification

In general, do not expect Linux FreeS/WAN to do everything yet. This is a work-in-progress and some parts of the IPsec specification are not yet implemented.

### In Linux FreeS/WAN

Things we do, as of version 1.96:

- key management methods
  - manually keyed*  
using keys stored in `/etc/ipsec.conf`
  - automatically keyed*  
Automatically negotiating session keys as required. All connections are automatically re-keyed periodically. The Pluto daemon implements this using the IKE protocol.
- Methods of authenticating gateways for IKE
  - shared secrets*  
stored in `ipsec.secrets(5)`
  - RSA signatures*  
For details, see `pluto(8)`.  
*looking up RSA authentication keys from DNS.*  
Note that this technique cannot be fully secure until secure DNS is widely deployed.
- groups for Diffie-Hellman key negotiation
  - group 2, modp 1024-bit*
  - group 5, modp 1536-bit*  
We implement these two groups.

In negotiating a keying connection (ISAKMP SA, Phase 1) we propose both groups when we are the initiator, and accept either when a peer proposes them. Once the keying connection is made, we propose only the alternative agreed there for data connections (IPsec SA's, Phase 2) negotiated over that keying connection.

- encryption transforms
  - DES*  
DES is in the source code since it is needed to implement 3DES, but single DES is not made available to users because DES is insecure.
  - Triple DES*  
implemented, and used as the default encryption in Linux FreeS/WAN.
- authentication transforms
  - HMAC using MD5*  
implemented, may be used in IKE or by AH or ESP transforms.
  - HMAC using SHA*  
implemented, may be used in IKE or by AH or ESP transforms.  
In negotiations, we propose both of these and accept either.

- compression transforms

### *IPComp*

IPComp as described in RFC 2393 was added for FreeS/WAN 1.6. Note that Pluto becomes confused if you ask it to do IPComp when the kernel cannot.

All combinations of implemented transforms are supported. Note that some form of packet-level ***authentication is required whenever encryption is used***. Without it, the encryption will not be secure.

## Deliberately omitted

We do not implement everything in the RFCs because some of those things are insecure. See our discussions of avoiding bogus security.

Things we deliberately omit which are required in the RFCs are:

- null encryption (to use ESP as an authentication-only service)
- single DES
- DH group 1, a 768-bit modp group

Since these are the only encryption algorithms and DH group the RFCs require, it is possible in theory to have a standards-conforming implementation which will not interoperate with FreeS/WAN. Such an implementation would be inherently insecure, so we do not consider this a problem.

Anyway, most implementations sensibly include more secure options as well, so dropping null encryption, single DES and Group 1 does not greatly hinder interoperability in practice.

We also do not implement some optional features allowed by the RFCs:

- aggressive mode for negotiation of the keying channel or ISAKMP SA. This mode is a little faster than main mode, but exposes more information to an eavesdropper.

In theory, this should cause no interoperability problems since all implementations are required to support the more secure main mode, whether or not they also allow aggressive mode.

In practice, it does sometimes produce problems with implementations such as Windows 2000 where aggressive mode is the default. Typically, these are easily solved with a configuration change that overrides that default.

## Not (yet) in Linux FreeS/WAN

Things we don't yet do, as of version 1.96:

- key management methods
  - ◆ authenticate key negotiations via local PKI server, but see links to user patches
  - ◆ authenticate key negotiations via secure DNS
  - ◆ unauthenticated key management, using Diffie-Hellman key agreement protocol without authentication. Arguably, this would be worth doing since it is secure against all passive attacks. On the other hand, it is vulnerable to an active man-in-the-middle attack.
- encryption transforms

## Introduction to FreeS/WAN

Currently Triple DES is the only encryption method Pluto will negotiate.

No additional encryption transforms are implemented, though the RFCs allow them and some other IPsec implementations support various of them. We are not eager to add more. See this FAQ question.

AES, the successor to the DES standard, is an excellent candidate for inclusion in FreeS/WAN, see links to user patches.

- authentication transforms

No optional additional authentication transforms are currently implemented. Likely SHA-256, SHA-384 and SHA-512 will be added when AES is.

- Policy checking on decrypted packets

To fully comply with the RFCs, it is not enough just to accept only packets which survive any firewall rules in place to limit what IPsec packets get in, and then pass KLIPS authentication. That is what FreeS/WAN currently does.

We should also apply additional tests, for example ensuring that all packets emerging from a particular tunnel have source and destination addresses that fall within the subnets defined for that tunnel, and that packets with those addresses that did not emerge from the appropriate tunnel are disallowed.

This will be done as part of a KLIPS rewrite. See these links and the design mailing list for discussion.

## Our PF-Key implementation

We use PF-key Version Two for communication between the KLIPS kernel code and the Pluto Daemon. PF-Key v2 is defined by RFC 2367.

The "PF" stands for Protocol Family. PF-Inet defines a kernel/userspace interface for the TCP/IP Internet protocols (TCP/IP), and other members of the PF series handle Netware, Appletalk, etc. PF-Key is just a PF for key-related matters.

### PF-Key portability

PF-Key came out of Berkeley Unix work and is used in the various BSD IPsec implementations, and in Solaris. This means there is some hope of porting our Pluto(8) to one of the BSD distributions, or of running their photurisd(8) on Linux if you prefer Photuris key management over IKE.

It is, however, more complex than that. The PK-Key RFC deliberately deals only with keying, not policy management. The three PF-Key implementations we have looked at — ours, OpenBSD and KAME — all have extensions to deal with security policy, and the extensions are different. There have been discussions aimed at sorting out the differences, perhaps for a version three PF-Key spec. All players are in favour of this, but everyone involved is busy and it is not clear whether or when these discussions might bear fruit.

## Kernels other than the latest 2.2.x and 2.4.y

We develop and test on Redhat Linux using the most recent kernel in the 2.2 and 2.4 series. In general, we

recommend you use the latest kernel in one of those series. Complications and caveats are discussed below.

### 2.0.x kernels

Consider upgrading to the 2.2 kernel series. If you want to stay with the 2.0 series, then we strongly recommend 2.0.39. Some useful security patches were added in 2.0.38.

Various versions of the code have run at various times on most 2.0.xx kernels, but the current version is only lightly tested on 2.0.39, and not at all on older kernels.

Some of our patches for older kernels are shipped in 2.0.37 and later, so they are no longer provided in FreeS/WAN. This means recent versions of FreeS/WAN will probably not compile on anything earlier than 2.0.37.

### 2.2 and 2.4 kernels

*FreeS/WAN 1.0*

ran only on 2.0 kernels

*FreeS/WAN 1.1 to 1.8*

ran on 2.0 or 2.2 kernels

ran on some development kernels, 2.3 or 2.4—test

*FreeS/WAN 1.9 to 1.96*

runs on 2.0, 2.2 or 2.4 kernels

In general, *we suggest the latest 2.2 kernel or 2.4 for production use.*

Of course no release can be guaranteed to run on kernels more recent than it is, so quite often there will be no stable FreeS/WAN for the absolute latest kernel. See the [FAQ](#) for discussion.

## Intel Linux distributions other than Redhat

We develop and test on Redhat 6.1 for 2.2 kernels, and on Redhat 7.1 or 7.2 for 2.4, so minor changes may be required for other distributions.

### Redhat 7.0

There are some problems with FreeS/WAN on Redhat 7.0. They are soluble, but we recommend you upgrade to a later Redhat instead..

Redhat 7 ships with two compilers.

- Their *gcc* is version 2.96. Various people, including the GNU compiler developers and Linus, have said fairly emphatically that using this was a mistake. 2.96 is a development version, not intended for production use. In particular, it will not compile a Linux kernel.
- Redhat therefore also ship a separate compiler, which they call *kgcc*, for compiling kernels.

Kernel Makefiles have *gcc* as a default, and must be adjusted to use *kgcc* before a kernel will compile on 7.0. This mailing list message gives details:

Subject: Re: AW: Installing IPsec on Redhat 7.0

## Introduction to FreeS/WAN

Date: Thu, 1 Feb 2001 14:32:52 -0200 (BRST)  
From: Mads Rasmussen <mads@cit.com.br>

> From [www.redhat.com/support/docs/gotchas/7.0/gotchas-7-6.html#ss6.1](http://www.redhat.com/support/docs/gotchas/7.0/gotchas-7-6.html#ss6.1)

cd to /usr/src/linux and open the Makefile in your favorite editor. You will need to look for a line similar to this:

```
CC = $(CROSS_COMPILE)gcc -D__KERNEL__ -I$(HPATH)
```

This line specifies which C compiler to use to build the kernel. It should be changed to:

```
CC = $(CROSS_COMPILE)kgcc -D__KERNEL__ -I$(HPATH)
```

for Red Hat Linux 7. The kgcc compiler is egcs 2.91.66. From here you can proceed with the typical compiling steps.

Check the [mailing list](#) archive for more recent news.

## SuSE Linux

SuSE 6.3 and later versions, at least in Europe, ship with FreeS/WAN included.

FreeS/WAN packages distributed for SuSE 7.0–7.2 were somehow miscompiled. You can find fixed packages on [Kurt Garloff's page](#).

Here are some notes for an earlier SuSE version.

### SuSE Linux 5.3

Date: Mon, 30 Nov 1998  
From: Peter Onion <ponion@srd.bt.co.uk>

... I got Saturdays snapshot working between my two SUSE5.3 machines at home.

The mods to the install process are quite simple. From memory and looking at the files on the SUSE53 machine here at work....

And extra link in each of the /etc/init.d/rc?.d directories called K35ipsec which SUSE use to shut a service down.

A few mods in /etc/init.d/ipsec to cope with the different places that SUSE put config info, and remove the incursion of /etc/rc.d/init.d/functions and . /etc/sysconfig/network as they don't exist and 1st one isn't needed anyway.

insert ". /etc/rc.config" to pick up the SUSE config info and use

```
if test -n "$NETCONFIG" -a "$NETCONFIG" != "YAST_ASK" ; then
```

to replace

```
[ ${NETWORKING} = "no" ] && exit 0
```

Create /etc/sysconfig as SUSE doesn't have one.

I think that was all (but I prob forgot something)....

## Introduction to FreeS/WAN

You may also need to fiddle initialisation scripts to ensure that `/var/run/pluto.pid` is removed when rebooting. If this file is present, Pluto does not come up correctly.

## Slackware

Subject: Re: linux-IPsec: Slackware distribution  
Date: Thu, 15 Apr 1999 12:07:01 -0700  
From: Evan Brewer <dmessiah@silcon.com>

> Very shortly, I will be needing to install IPsec on at least gateways that  
> are running Slackware. . . .

The only trick to getting it up is that on the slackware dist there is no init.d directory in /etc/rc.d .. so create one. Then, what I do is take the IPsec startup script which normally gets put into the init.d directory, and put it in /etc/rc.d and name it rc.ipsec .. then I symlink it to the file in init.d. The only file in the dist you need to really edit is the utils/Makefile, setup4:

Everything else should be just fine.

A year or so later:

Subject: Re: HTML Docs- Need some cleanup?  
Date: Mon, 8 Jan 2001  
From: Jody McIntyre <jodym@oeone.com>

I have successfully installed FreeS/WAN on several Slackware 7.1 machines. FreeS/WAN installed its rc.ipsec file in /etc/rc.d. I had to manually call this script from rc.inet2. This seems to be an easier method than Evan Brewer's.

## Debian

A recent (Nov 2001) mailing list points to a [web page](#) on setting up several types of tunnel, including IPsec, on Debian.

Some older information:

Subject: FreeS/WAN 1.0 on Debian 2.1  
Date: Tue, 20 Apr 1999  
From: Tim Miller <cerebus+counterpane@haybaler.sackheads.org>

Compiled and installed without error on a Debian 2.1 system with kernel-source-2.0.36 after pointing RCDIR in utils/Makefile to /etc/init.d.

/var/lock/subsys/ doesn't exist on Debian boxen, needs to be created; not a fatal error.

Finally, IPsec scripts appear to be dependant on GNU awk (gawk); the default Debian awk (mawk-1.3.3-2) had fatal difficulties. With gawk installed and /etc/alternatives/awk linked to /usr/bin/gawk operation appears flawless.

The scripts in question have been modified since this was posted. Awk versions should no longer be a problem.

## Caldera

Subject: Re: HTML Docs- Need some cleanup?  
Date: Mon, 08 Jan 2001  
From: Andy Bradford <andyb@calderasystems.com>

On Sun, 07 Jan 2001 22:59:05 EST, Sandy Harris wrote:

```
> Intel Linux distributions other than Redhat 5.x and 6.x
> Redhat 7.0
> SuSE Linux
> SuSE Linux 5.3
> Slackware
> Debian
```

Can you please include Caldera in this list? I have tested it since FreeS/Wan 1.1 and it works great with our systems---provided one follows the FreeS/Wan documentation. :-)

Thank you,  
Andy

## CPUs other than Intel

FreeS/WAN has been run successfully on a number of different CPU architectures. If you have tried it on one not listed here, please post to the [mailing list](#).

## Corel Netwinder (StrongARM CPU)

Subject: linux-ipsec: Netwinder diffs  
Date: Wed, 06 Jan 1999  
From: rhatfield@plaintree.com

I had a mistake in my IPsec-auto, so I got things working this morning.

Following are the diffs for my changes. Probably not the best and cleanest way of doing it, but it works. . . .

These diffs are in the 0.92 and later distributions, so these should work out-of-the-box on Netwinder.

## Yellow Dog Linux on Power PC

Subject: Compiling FreeS/WAN 1.1 on YellowDog Linux (PPC)  
Date: 11 Dec 1999  
From: Darron Froese <darron@fudgehead.com>

I'm summarizing here for the record - because it's taken me many hours to do this (multiple times) and because I want to see IPsec on more linuxes than just x86.

Also, I can't remember if I actually did summarize it before... ;-) I'm working too many late hours.

That said - here goes.

1. Get your linux kernel and unpack into /usr/src/linux/ - I used 2.2.13.  
<<http://www.kernel.org/pub/linux/kernel/v2.2/linux-2.2.13.tar.bz2>>

## Introduction to FreeS/WAN

2. Get FreeS/WAN and unpack into /usr/src/freeswan-1.1  
<ftp://ftp.xs4all.nl/pub/crypto/freeswan/freeswan-1.1.tar.gz>

3. Get the gmp src rpm from here:  
<ftp://ftp.yellowdoglinux.com/pub/yellowdog/champion-1.1/SRPMS/SRPMS/gmp-2.0.2-9a.src.rpm>

4. Su to root and do this: rpm --rebuild gmp-2.0.2-9a.src.rpm

You will see a lot of text fly by and when you start to see the rpm recompiling like this:

```
Executing: %build
+ umask 022
+ cd /usr/src/redhat/BUILD
+ cd gmp-2.0.2
+ libtoolize --copy --force
Remember to add `AM_PROG_LIBTOOL' to `configure.in'.
You should add the contents of `/usr/share/aclocal/libtool.m4' to
`aclocal.m4'.
+ CFLAGS=-O2 -fsigned-char
+ ./configure --prefix=/usr
```

Hit Control-C to stop the rebuild. NOTE: We're doing this because for some reason the gmp source provided with FreeS/WAN 1.1 won't build properly on ydl.

```
cd /usr/src/redhat/BUILD/
cp -ar gmp-2.0.2 /usr/src/freeswan-1.1/
cd /usr/src/freeswan-1.1/
rm -rf gmp
mv gmp-2.0.2 gmp
```

5. Open the freeswan Makefile and change the line that says:  
KERNEL=\$(b)zimage (or something like that) to  
KERNEL=vmlinux

6. cd ../linux/

7. make menuconfig  
Select an option or two and then exit - saving your changes.

8. cd ../freeswan-1.1/ ; make menugo

That will start the whole process going - once that's finished compiling, you have to install your new kernel and reboot.

That should build FreeS/WAN on ydl (I tried it on 1.1).

And a later message on the same topic:

Subject: Re: FreeS/WAN, PGPnet and E-mail  
Date: Sat, 22 Jan 2000  
From: Darron Froese <darron@fudgehead.com>

on 1/22/00 6:47 PM, Philip Trauring at philip@trauring.com wrote:

> I have a PowerMac G3 ...

The PowerMac G3 can run YDL 1.1 just fine. It should also be able to run FreeS/WAN 1.2patch1 with a couple minor modifications:

# Introduction to FreeS/WAN

1. In the Makefile it specifies a bzipimage for the kernel compile - you have to change that to vmlinux for the PPC.

2. The gmp source that comes with FreeS/WAN (for whatever reason) fails to compile. I have gotten around this by getting the gmp src rpm from here:

```
ftp://ftp.yellowdoglinux.com/pub/yellowdog/champion-1.1/SRPMS/SRPMS/gmp-2.0.2-9a.src.rpm
```

If you rip the source out of there - and place it where the gmp source resides it will compile just fine.

FreeS/WAN no longer includes GMP source.

## Mklinux

One user reports success on the Mach-based *microkernel* Linux.

Subject: Smiles on sparc and ppc  
Date: Fri, 10 Mar 2000  
From: Jake Hill <jah@alien.bt.co.uk>

You may or may not be interested to know that I have successfully built FreeS/WAN on a number of non intel alpha architectures; namely on ppc and sparc and also on osfmach3/ppc (MkLinux). I can report that it just works, mostly, with few changes.

## Alpha 64-bit processors

Subject: IT WORKS (again) between intel & alpha :-))))  
Date: Fri, 29 Jan 1999  
From: Peter Onion <ponion@srd.bt.co.uk>

Well I'm happy to report that I've got an IPsec connection between by intel & alpha machines again

If you look back on this list to 7th of December I wrote...

```
-On 07-Dec-98 Peter Onion wrote:
->
-> I've about had enuf of wandering around inside the kernel trying to find out
-> just what is corrupting outgoing packets...
-
-Its 7:30 in the evening .....
-
-I FIXED IT :-))))))))))))))))))))))))))))))))))))))
-
-It was my own fault :-(((((((((((((((((((((
-
-If you ask me very nicly I'll tell you where I was a little too over keen to
-change unsigned long int __u32 :-) OPSE ...
-
-So tomorrow it will full steam ahead to produce a set of diffs/patches against
-0.91
-
-Peter Onion.
```

In general (there have been some glitches), FreeS/WAN has been running on Alphas since then.

## Sun SPARC processors

Several users have reported success with FreeS/WAN on SPARC Linux. Here is one mailing list message:

Subject: Smiles on sparc and ppc  
Date: Fri, 10 Mar 2000  
From: Jake Hill <jah@alien.bt.co.uk>

You may or may not be interested to know that I have successfully built FreeS/WAN on a number of non intel alpha architectures; namely on ppc and sparc and also on osfmach3/ppc (MkLinux). I can report that it just works, mostly, with few changes.

I have a question, before I make up some patches. I need to hack gmp/mpn/powerpc32/\*.s to build them. Is this ok? The changes are trivial, but could I also use a different version of gmp? Is it vanilla here?

I guess my only real headache is from ipchains, which appears to stop running when IPsec has been started for a while. This is with 2.2.14 on sparc.

This message, from a different mailing list, may be relevant for anyone working with FreeS/WAN on Suns:

Subject: UltraSPARC DES assembler  
Date: Thu, 13 Apr 2000  
From: svolaf@inet.uni2.dk (Svend Olaf Mikkelsen)  
To: coderpunks@toad.com

An UltraSPARC assembler version of the LibDES/SSLeay/OpenSSL des\_enc.c file is available at <http://inet.uni2.dk/~svolaf/des.htm>.

This brings DES on UltraSPARC from slower than Pentium at the same clock speed to significantly faster.

## MIPS processors

We know FreeS/WAN runs on at least some MIPS processors because Lasat manufacture an IPsec box based on an embedded MIPS running Linux with FreeS/WAN. We have no details.

## Transmeta Crusoe

The Merilus Firecard, a Linux firewall on a PCI card, is based on a Crusoe processor and supports FreeS/WAN.

## Motorola Coldfire

Subject: Re: Crypto hardware support  
Date: Mon, 03 Jul 2000  
From: Dan DeVault <devault@tampabay.rr.com>

.... I have been running uClinux with FreeS/WAN 1.4 on a system built by Moreton Bay ( <http://www.moretonbay.com> ) and it was using a Coldfire processor and was able to do the Triple DES encryption at just about 1 mbit / sec rate..... they put a Hi/Fn 7901 hardware encryption

chip on their board and now their system does over 25 mbit of 3DES encryption..... pretty significant increase if you ask me.

## Multiprocessor machines

FreeS/WAN is designed to work on SMP (symmetric multi-processing) Linux machines and is regularly tested on dual processor x86 machines.

We do not know of any testing on multi-processor machines with other CPU architectures or with more than two CPUs. Anyone who does test this, please report results to the [mailing list](#).

The current design does not make particularly efficient use of multiprocessor machines; some of the kernel work is single-threaded.

## Support for crypto hardware

Supporting hardware cryptography accelerators has not been a high priority for the development team because it raises a number of fairly complex issues:

- Can you trust the hardware? If it is not Open Source, how do you audit its security? Even if it is, how do you check that the design has no concealed traps?
- If an interface is added for such hardware, can that interface be subverted or misused?
- Is hardware acceleration actually a performance win? It clearly is in many cases, but on a fast machine it might be better to use the CPU for the encryption than to pay the overheads of moving data to and from a crypto board.
- the current KLIPS code does not provide a clean interface for hardware accelerators

That said, we have a [report](#) of FreeS/WAN working with one crypto accelerator and some work is going on to modify KLIPS to create a clean generic interface to such products. See this [web page](#) for some of the design discussion.

More recently, a patch to support some hardware accelerators has been posted:

```
Subject: [Design] [PATCH] H/W acceleration patch
Date: Tue, 18 Sep 2001
From: "Martin Gadbois" <martin.gadbois@colubris.com>
```

Finally!!

Here's a web site with H/W acceleration patch for FreeS/WAN 1.91, including S/W and Hifn 7901 crypto support.

<http://sources.colubris.com/>

Martin Gadbois

Hardware accelerators could take performance well beyond what FreeS/WAN can do in software (discussed [here](#)). Here is some discussion off the IETF IPsec list, October 2001:

```
... Currently shipping chips deliver, 600 mbps throughput on a single
stream of 3DES IPsec traffic. There are also chips that use multiple
cores to do 2.4 gbps. We (Cavium) and others have announced even faster
chips. ... Mid 2002 versions will handle at line rate (OC48 and OC192)
IPsec and SSL/TLS traffic not only 3DES CBC but also AES and arc4.
```

The patches to date support chips that have been in production for some time, not the state-of-the-art latest-and-greatest devices described in that post. However, they may still outperform software and they almost certainly reduce CPU overhead.

## IP version 6 (IPng)

The Internet currently runs on version four of the IP protocols. IPv4 is what is in the standard Linux IP stack, and what FreeS/WAN was built for. In IPv4, IPsec is an optional feature.

The next version of the IP protocol suite is version six, usually abbreviated either as "IPv6" or as "IPng" for "IP: the next generation". For IPv6, IPsec is a required feature. Any machine doing IPv6 is required to support IPsec, much as any machine doing (any version of) IP is required to support ICMP.

There is a Linux implementation of IPv6 in Linux kernels 2.2 and above. For details, see the [FAQ](#). It does not yet support IPsec. The [USAGI](#) project are also working on IPv6 for Linux.

FreeS/WAN was originally built for the current standard, IPv4, but we are interested in seeing it work with IPv6. Some progress has been made, and a patched version with IPv6 support is [available](#). For more recent information, check the [mailing list](#).

## IPv6 background

IPv6 has been specified by an IETF [working group](#). The group's page lists over 30 RFCs to date, and many Internet Drafts as well. The overview is [RFC 2460](#). Major features include:

- expansion of the address space from 32 to 128 bits,
- changes to improve support for
  - ◆ mobile IP
  - ◆ automatic network configuration
  - ◆ quality of service routing
  - ◆ ...
- improved security via IPsec

A number of projects are working on IPv6 implementation. A prominent Open Source effort is [KAME](#), a collaboration among several large Japanese companies to implement IPv6 for Berkeley Unix. Other major players are also working on IPv6. For example, see pages at:

- [Sun](#)
- [Cisco](#)
- [Microsoft](#)

The [6bone](#) (IPv6 backbone) testbed network has been up for some time. There is an active [IPv6 user group](#).

One of the design goals for IPv6 was that it must be possible to convert from v4 to v6 via a gradual transition process. Imagine the mess if there were a "flag day" after which the entire Internet used v6, and all software designed for v4 stopped working. Almost every computer on the planet would need major software changes! There would be huge costs to replace older equipment. Implementers would be worked to death before "the day", systems administrators and technical support would be completely swamped after it. The bugs in every implementation would all bite simultaneously. Large chunks of the net would almost certainly be down for substantial time periods. ...

## Introduction to FreeS/WAN

Fortunately, the design avoids any "flag day". It is therefore a little tricky to tell how quickly IPv6 will take over. The transition has certainly begun. For examples, see announcements from [NTT](#) and [Nokia](#). However, it is not yet clear how quickly the process will gain momentum, or when it will be completed. Likely large parts of the Internet will remain with IPv4 for years to come.

# Interoperating with FreeS/WAN

The FreeS/WAN project needs you! We rely on the user community to keep up to date. Mail [users@lists.freeswan.org](mailto:users@lists.freeswan.org) with your interop success stories.

**Please note:** Most of our interop examples feature Linux FreeS/WAN 1.x config files. You can convert them to 2.x files fairly easily with the patch in our [Upgrading Guide](#).

## Interop at a Glance

	FreeS/WAN VPN					Road Warrior	OE
	PSK	RSA Secret	X.509 (requires patch)	NAT-Traversal (requires patch)	Manual Keying		
More Compatible							
<a href="#">FreeS/WAN</a>	Yes	Yes	Yes	Yes	Yes	Yes	Yes
<a href="#">isakmpd (OpenBSD)</a>	Yes		Yes		Yes		No
<a href="#">Kame (FreeBSD, NetBSD, MacOSX)</a> <small>aka racoon</small>	Yes	Yes	Yes		Yes		No
<a href="#">McAfee VPN</a> <small>was PGPNet</small>	Yes	Yes	Yes			Yes	No
<a href="#">Microsoft Windows 2000/XP</a>	Yes		Yes			Yes	No
<a href="#">SSH Sentinel</a>	Yes		Yes	Maybe		Yes	No
<a href="#">Safenet SoftPK /SoftRemote</a>	Yes		Yes			Yes	No
Other							
<a href="#">6Wind</a>			Yes				No
<a href="#">Alcatel Timestep</a>	Yes						No
<a href="#">Apple Macintosh System 10+</a>	Maybe	Yes	Maybe		Maybe		No
<a href="#">AshleyLaurent VPCOM</a>	Yes						No
<a href="#">Borderware</a>	Yes					No	No
<a href="#">Check Point FW-1/VPN-1</a>	Yes		Yes			Yes	No
<a href="#">Cisco with 3DES</a>	Yes	Maybe		Maybe			No
<a href="#">Equinux VPN Tracker (for Mac OS X)</a>	Yes	Yes	Yes		Maybe		No
<a href="#">F-Secure</a>	Yes			Maybe	Yes	Yes	No
<a href="#">Gauntlet GVPN</a>	Yes		Yes				No
<a href="#">IBM AIX</a>	Yes		Maybe				No
<a href="#">IBM AS/400</a>	Yes						No

## Introduction to FreeS/WAN

<u>Intel Shiva</u>	Yes						No
<u>LANRover/Net Structure</u>	Yes						No
<u>LanCom (formerly ELSA)</u>	Yes						No
<u>Linksys</u>	Maybe		No			Yes	No
<u>Lucent</u>	Partial						No
<u>Netasq</u>			Yes				No
<u>netcelo</u>			Yes				No
<u>Netgear fvs318</u>	Yes						No
<u>Netscreen 100</u> <u>or 5xp</u>	Yes					Maybe	No
<u>Nortel Contivity</u>	Partial		Yes	Maybe			No
<u>RadGuard</u>	Yes						No
<u>Raptor</u>	Yes				Yes		No
<u>Redcreek Ravlin</u>	Yes/Partial						No
<u>SonicWall</u>	Yes				Maybe	No	No
<u>Sun Solaris</u>	Yes		Yes		Yes		No
<u>Symantec</u>	Yes						No
<u>Watchguard</u> <u>Firebox</u>	Yes				Yes		No
<u>Xedia Access Point</u> <u>/QVPN</u>	Yes						No
<u>Zyxel Zywall</u> <u>/Prestige</u>	Yes						No
	PSK	RSA Secret	X.509 (requires patch)	NAT-Traversal (requires patch)	Manual Keying		
	FreeS/WAN VPN					Road Warrior	OE

### Key

Yes	People report that this works for them.
[Blank]	We don't know.
No	We have reason to believe it was, at some point, not possible to get this to work.
Partial	Partial success. For example, a connection can be created from one end only.
Yes/Partial	Mixed reports.
Maybe	We think the answer is "yes", but need confirmation.

## Basic Interop Rules

Vanilla FreeS/WAN implements these parts of the IPSec specifications. You can add more with Super FreeS/WAN, but what we offer may be enough for many users.

- To use X.509 certificates with FreeS/WAN, you will need the X.509 patch or Super FreeS/WAN, which includes that patch.

## Introduction to FreeS/WAN

- To use [Network Address Translation](#) (NAT) traversal with FreeS/WAN, you will need Arkoon Network Security's [NAT traversal patch](#) or [Super FreeS/WAN](#), which includes it.

We offer a set of proposals which is not user-adjustable, but covers all combinations that we can offer. FreeS/WAN always proposes triple DES encryption and Perfect Forward Secrecy (PFS). In addition, we propose Diffie Hellman groups 5 and 2 (in that order), and MD5 and SHA-1 hashes. We accept the same proposals, in the same order of preference.

Other interop notes:

- A [SHA-1 bug in FreeS/WAN 2.00, 2.01 and 2.02](#) may affect some interop scenarios. It does not affect 1.x versions, and is fixed in 2.03 and later.
- Some other implementations will close a connection with FreeS/WAN after some time. This may be a problem with rekey lifetimes. Please see [this tip](#) and [this workaround](#).

## Longer Stories

### For *More Compatible* Implementations

#### FreeS/WAN

See our documentation at [freeswan.org](http://freeswan.org) and the Super FreeS/WAN docs at [freeswan.ca](http://freeswan.ca). Some user-written HOWTOs for FreeS/WAN-FreeS/WAN connections are listed in [our Introduction](#).

See also:

- [A German FreeS/WAN-FreeS/WAN page by Markus Wernig \(X.509\)](#)

[Back to chart](#)

#### isakmpd (OpenBSD)

[OpenBSD FAQ: Using IPsec](#)

[Hans-Joerg Hoexer's interop Linux-OpenBSD \(PSK\)](#)

[Skyper's configuration \(PSK\)](#)

[French page with configs \(X.509\)](#)

[Back to chart](#)

#### Kame

- For FreeBSD and NetBSD. Ships with Mac OS X; see also our [Mac](#) section.
- Also known as *racoona*, its keying daemon.

[Kame homepage, with FAQ](#)

[NetBSD's IPSec FAQ](#)

[Ghislaine's post explaining some interop peculiarities](#)

[Itojun's Kame-FreeS/WAN interop tips \(PSK\)](#)

[Ghislaine Labouret's French page with links to matching FreeS/WAN and Kame configs \(RSA\)](#)

## Introduction to FreeS/WAN

[Markus Wernig's HOWTO \(X.509, BSD gateway\)](#)

[Frodo's Kame-FreeS/WAN interop \(X.509\)](#)

[Kame as a WAVEsec client.](#)

[Back to chart](#)

### PGPNet/McAfee

- Now called McAfee VPN Client.
- PGPNet also came in a freeware version which did not support subnets
- To support dhcp-over-ipsec, you need the X.509 patch, which is included in [Super FreeS/WAN](#).

[Tim Carr's Windows Interop Guide \(X.509\)](#)

[Hans-Joerg Hoexer's Guide for Linux-PGPNet \(PSK\)](#)

[Kai Martius' instructions using RSA Key-Extractor Tool \(RSA\)](#)

[Christian Zeng's page \(RSA\)](#) based on Kai's work. English or German.

[Oscar Delgado's PDF \(X.509, no configs\)](#)

[Ryan's HOWTO for FreeS/WAN-PGPNet \(X.509\)](#). Through a Linksys Router with IPsec Passthru enabled.

[Jean-Francois Nadeau's Practical Configuration \(Road Warrior with PSK\)](#)

[Wouter Prins' HOWTO \(Road Warrior with X.509\)](#)

[Rekeying problem with FreeS/WAN and older PGPnets](#)

[DHCP over IPSEC HOWTO for FreeS/WAN \(requires X.509 and dhcprelay patches\)](#)

[Back to chart](#)

### Microsoft Windows 2000/XP

- IPsec comes with Win2k, and with XP Support Tools. May require [High Encryption Pack](#). WinXP users have also reported better results with Service Pack 1.
- The Road Warrior setup works either way round. Windows (XP or 2K) IPsec can connect as a Road Warrior to FreeS/WAN. However, FreeS/WAN can also successfully connect as a Road Warrior to Windows IPsec (see Nate Carlson's configs below).
- FreeS/WAN version 1.92 or later is required to avoid an interoperation problem with Windows native IPsec. Earlier FreeS/WAN versions did not process the Commit Bit as Windows native IPsec expected.

[Tim Carr's Windows Interop Guide \(X.509\)](#)

[James Carter's instructions \(X.509, NAT-T\)](#)

[Jean-Francois Nadeau's Net-net Configuration \(PSK\)](#)

[Telenor's Node-node Config \(Transport-mode PSK\)](#)

[Marcus Mueller's HOWTO using his VPN config tool \(X.509\)](#). Tool also works with PSK.

[Nate Carlson's HOWTO using same tool \(Road Warrior with X.509\)](#). Unusually, FreeS/WAN is the Road Warrior here.

[Oscar Delgado's PDF \(X.509, no configs\)](#)

[Tim Scannell's Windows XP Additional Checklist \(X.509\)](#)

[Microsoft's page on Win2k TCP/IP security features](#)

[Microsoft's Win2k IPsec debugging tips](#)

[MS VPN may fall back to 1DES](#)

[Back to chart](#)

## SSH Sentinel

- Popular and well tested.
- Also rebranded in [Zyxel Zywall](#). Our Zyxel interop notes are [here](#).
- SSH supports IPsec-over-UDP NAT traversal.
- There is this [potential problem](#) if you're not using the Legacy Proposal option.

[SSH's Sentinel-FreeSWAN interop PDF \(X.509\)](#)

[Nadeem Hassan's SUSE-to-Sentinel article \(Road warrior with X.509\)](#)

[O-Zone's Italian HOWTO \(Road Warrior, X.509, DHCP\)](#)

[Back to chart](#)

## Safenet SoftPK/SoftRemote

- People recommend SafeNet as a low cost Windows client.
- SoftRemote seems to be the newer name for SoftPK.

[Whit Blauvelt's SoftRemote tips](#)

[Tim Wilson's tips \(X.509\) Workaround for a "gotcha"](#)

[Jean-Francois Nadeau's Practical Configuration \(Road Warrior with PSK\)](#)

[Terradon Communications' PDF \(Road Warrior with PSK\)](#)

[Seaan.net's PDF \(Road Warrior to Subnet, with PSK\)](#)

[Red Baron Consulting's PDF \(Road Warrior with X.509\)](#)

[Back to chart](#)

## For *Other Implementations*

### 6Wind

[French page with configs \(X.509\)](#)

[Back to chart](#)

### Alcatel Timestep

[Alain Sabban's settings \(PSK or PSK road warrior; through static NAT\)](#)

[Derick Cassidy's configs \(PSK\)](#)

[David Kerry's Timestep settings \(PSK\)](#)

[Kevin Gerbracht's ipsec.conf \(X.509\)](#)

[Back to chart](#)

## Apple Macintosh System 10+

- Since the system is based on FreeBSD, this should interoperate just like FreeBSD.
- To use Appletalk over IPsec tunnels, run it over TCP/IP, or use Open Door Networks' Shareway IP tool, described here.
- See also the Equinux VPN Tracker for Mac OS X.

James Carter's instructions (X.509, NAT-T)

Back to chart

## AshleyLaurent VPCom

Successful interop report, no details

Back to chart

## Borderware

- I suspect the Borderware client is a rebranded Safenet. If that's true, our Safenet section will help.

Philip Reetz' configs (PSK)

Borderware server does not support FreeS/WAN road warriors

Older Borderware may not support Diffie Hellman groups 2, 5

Back to chart

## Check Point VPN-1 or FW-1

- Caveat about IP-range inclusion on Check Point.
- Some versions of Check Point may require an aggressive mode patch to interoperate with FreeS/WAN.  
Super FreeS/WAN now features this patch.
- A Linux FreeS/WAN-Checkpoint connection may close after some time. Try this tip toward a workaround.

AERAssec's Firewall-1 NG site (PSK, X.509, Road Warrior with X.509, other algorithms)

AERAssec's detailed Check Point-FreeS/WAN support matrix

Checkpoint.com PDF: Linux as a VPN Client to FW-1 (PSK)

PhoneBoy's Check Point FAQ (on Check Point only, not FreeS/WAN)

Chris Harwell's tips & FreeS/WAN configs (PSK)

Daniel Tombeil's configs (PSK)

Back to chart

## Cisco

- Cisco supports IPsec-over-UDP NAT traversal.
- Cisco VPN Client appears to use nonstandard IPsec and does not work with FreeS/WAN. This message concerns Cisco VPN Client 4.01.

## Introduction to FreeS/WAN

- A Linux FreeS/WAN–Cisco connection may close after some time. [Here](#) is a workaround, and [here](#) is another comment on the same subject.
- [Older Ciscos](#) purchased outside the United States may not have 3DES, which FreeS/WAN requires.
- [RSA keying may not be possible between Cisco and FreeS/WAN.](#)
- [In ipsec.conf, VPN3000 DN \(distinguished name\) must be in binary \(X.509 only\)](#)

[SANS Institute HOWTO \(PSK\)](#). Detailed, with extensive references.

[Short HOWTO \(PSK\)](#)

[French page with configs for Cisco IOS, PIX and VPN 3000 \(X.509\)](#)

[Dave McFerren's sample configs \(PSK\)](#)

[Wolfgang Tremmel's sample configs \(PSK road warrior\)](#)

[Old doc from Pete Davis, with William Watson's updated Tips \(PSK\)](#)

### *Some PIX specific information:*

[Waikato Linux Users' Group HOWTO. Nice detail \(PSK\)](#)

[John Leach's configs \(PSK\)](#)

[Greg Robinson's settings \(PSK\)](#)

[Scott's ipsec.conf for PIX \(PSK, FreeS/WAN side only\)](#)

[Rick Trimble's PIX and FreeS/WAN settings \(PSK\)](#)

[Cisco VPN support page](#)

[Cisco IPsec information page](#)

[Back to chart](#)

## Equinux VPN tracker (for Mac OS X)

- Graphical configurator for Mac OS X IPsec. May be an interface to the [native Mac OS X IPsec](#), which is essentially [KAME](#).
- To use Appletalk over IPsec tunnels, [run it over TCP/IP](#), or use Open Door Networks' Shareway IP tool, [described here](#).

Equinux provides [this excellent interop PDF](#) (PSK, RSA, X.509).

[Back to chart](#)

## F–Secure

- F–Secure supports IPsec–over–UDP NAT traversal.

[pingworks.de's "Connecting F–Secure's VPN+ to Linux FreeS/WAN" \(PSK road warrior\)](#)

[Same thing as PDF](#)

[Success report, no detail \(PSK\)](#)

[Success report, no detail \(Manual\)](#)

[Back to chart](#)

## Gauntlet GVPN

[Richard Reiner's ipsec.conf \(PSK\)](#)

[Might work without that pesky firewall... \(PSK\)](#)

## Introduction to FreeS/WAN

In late July, 2003 Alexandar Antik reported success interoperating with Gauntlet 6.0 for Solaris (X.509). Unfortunately the message is not properly archived at this time.

[Back to chart](#)

### IBM AIX

[IBM's "Built-In Network Security with AIX" \(PSK, X.509\)](#)

[IBM's tip: importing Linux FreeS/WAN settings into AIX's \*ikedb\* \(PSK\)](#)

[Back to chart](#)

### IBM AS/400

- [Road Warriors may act flaky.](#)

[Richard Welty's tips and tricks](#)

[Back to chart](#)

### Intel Shiva LANRover / Net Structure

- Intel Shiva LANRover is now known as Intel Net Structure.
- [Shiva seems to have two modes: IPsec or the proprietary "Shiva Tunnel".](#) Of course, FreeS/WAN will only create IPsec tunnels.
- [AH may not work for Shiva-FreeS/WAN.](#) That's OK, since FreeS/WAN has phased out the use of AH.

[Snowcrash's configs \(PSK\)](#)

[Old configs from an interop \(PSK\)](#)

[The day Shiva tickled a Pluto bug \(PSK\)](#)

[Follow up: success!](#)

[Back to chart](#)

### LanCom (formerly ELSA)

- This router is popular in Germany.

Jakob Curdes successfully created a PSK connection with the LanCom 1612 in August 2003.

[Back to chart](#)

### Linksys

- Linksys may be used as an IPsec tunnel endpoint, **OR** as a router in "IPsec passthrough" mode, so that the IPsec tunnel passes through the Linksys.

### As tunnel endpoint

[Ken Bantoft's instructions \(Road Warrior with PSK\)](#)

[Nate Carlson's caveats](#)

### In IPsec passthrough mode

[Sample HOWTO through a Linksys Router](#)

[Nadeem Hasan's configs](#)

[Brock Nanson's tips](#)

[Back to chart](#)

### Lucent

[Partial success report; see also the next message in thread](#)

[Back to chart](#)

### Netasq

[French page with configs \(X.509\)](#)

[Back to chart](#)

### Netcelo

[French page with configs \(X.509\)](#)

[Back to chart](#)

### Netgear fvs318

- With a recent Linux FreeS/WAN, you will require the latest (12/2002) Netgear firmware, which supports Diffie–Hellman (DH) group 2. For security reasons, we phased out DH 1 after Linux FreeS/WAN 1.5.
- [This message](#) reports the incompatibility between Linux FreeS/WAN 1.6+ and Netgear fvs318 without the firmware upgrade.
- We believe Linux FreeS/WAN 1.5 and earlier will interoperate with any NetGear firmware.

[John Morris' setup \(PSK\)](#)

[Back to chart](#)

### Netscreen 100 or 5xp

[Errol Neal's settings \(PSK\)](#)

[Corey Rogers' configs \(PSK, no PFS\)](#)

[Jordan Share's configs \(PSK, 2 subnets, through static NAT\)](#)

[Set src proxy\\_id to your protected subnet/mask](#)

[French page with ipsec.conf, Netscreen screen shots \(X.509, may need to revert to PSK...\)](#)

## Introduction to FreeS/WAN

[A report of a company using Netscreen with FreeS/WAN on a large scale \(FreeS/WAN road warriors?\)](#)

[Back to chart](#)

### Nortel Contivity

- Nortel supports IPsec-over-UDP NAT traversal.
- Some older versions of Contivity and FreeS/WAN will not communicate.
- FreeS/WAN cannot be used as a "client" to a Nortel Contivity server, but can be used as a branch-office tunnel.
- Contivity does not send Distinguished Names in the order FS wants them (X.509).
- Connections may time out after 30–40 minutes idle.

[JJ Streicher–Bremer's mini HOWTO for old & new software. \(PSK with two subnets\)](#)  
[French page with configs \(X.509\).](#) This succeeds using the above X.509 tip.

[Back to chart](#)

### Radguard

[Marko Hausalo's configs \(PSK\).](#) Note: These do create a connection, as you can see by "IPsec SA established".

[Claudia Schmeing's comments](#)

[Back to chart](#)

### Raptor (NT or Solaris)

- Now known as Symantec Enterprise Firewall.
- The Raptor does not normally come with X.509, but this may be available as an add-on.
- Raptor requires alphanumeric PSK values, whereas FreeS/WAN uses hex.
- Raptor's tunnel endpoint may be a host, subnet or group of subnets (see [this message](#) ). FreeS/WAN cannot handle the group of subnets; you must create separate connections for each in order to interoperate.
- Some versions of Raptor accept only single DES. According to this German message, the Raptor Mobile Client demo offers single DES only.

[Peter Mazinger's settings \(PSK\)](#)

[Peter Gerland's configs \(PSK\)](#)

[Charles Griebel's configs \(PSK\).](#)

[Lumir Srch's tips \(PSK\)](#)

[John Hardy's configs \(Manual\)](#)

[Older Raptors want 3DES keys in 3 parts \(Manual\).](#)

[Different keys for each direction? \(Manual\)](#)

[Back to chart](#)

## Redcreek Ravlin

- Known issue #1: The Ravlin expects a quick mode renegotiation right after every Main Mode negotiation.
- Known issue #2: The Ravlin tries to negotiate a zero connection lifetime, which it takes to mean "infinite". [Jim Hague's patch](#) addresses both issues.
- [Interop works with Ravlin Firmware > 3.33. Includes tips \(PSK\).](#)

[Back to chart](#)

## SonicWall

- [Sonicwall cannot be used for Road Warrior setups](#)
- At one point, [only Sonicwall PRO supported triple DES](#).
- [Older Sonicwalls \(before Nov 2001\) feature Diffie Hellman group 1 only.](#)

[Paul Wouters' config \(PSK\)](#)

[Dilan Arumainathan's configuration \(PSK\)](#)

[Dariush's setup... only opens one way \(PSK\)](#)

[Andreas Steffen's tips \(X.509\)](#)

[Back to chart](#)

## Sun Solaris

- Solaris 8+ has a native (in kernel) IPsec implementation.
- [Solaris does not seem to support tunnel mode, but you can make IP-in-IP tunnels instead, like this.](#)

[Reports of some successful interops](#) from a fellow @sun.com. See also [these follow up posts](#).

[Aleks Shenkman's configs \(Manual in transport mode\)](#)

[Back to chart](#)

## Symantec

- The Raptor, covered [above](#), is now known as Symantec Enterprise Firewall.
- Symantec's "distinguished name" is a KEY\_ID. See Andreas Steffen's post, below.

[Andreas Steffen's configs for Symantec 200R \(PSK\)](#)

[Back to chart](#)

## Watchguard Firebox

- Automatic keying works with WatchGuard 5.0+ only.
- Seen to interoperate with WatchGuard 1000, II, III; firmware v. 5, 6..
- For manual keying, Watchguard's Policy Manager expects SPI numbers and encryption and authentication keys in decimal (not hex).

[WatchGuard's HOWTO \(PSK\)](#)

[Ronald C. Riviera's Settings \(PSK\)](#)

[Walter Wickersham's Notes \(PSK\)](#)

[Max Enders' Configs \(Manual\)](#)

[Old known issue with auto keying](#)

[Tips on key generation and format \(Manual\)](#)

[Back to chart](#)

## **Xedia Access Point/QVPN**

[Hybrid IPsec/L2TP connection settings \(X.509\)](#)

[Xedia's LAN-LAN links don't use multiple tunnels](#)

[That explanation, continued](#)

[Back to chart](#)

## **Zyxel**

- The Zyxel Zywall is a rebranded SSH Sentinel box. See also our section on [SSH](#).
- There seems to be a problem with keeping this connection alive. This is caused at the Zyxel end. See this brief [discussion and solution](#).

[Zyxel's Zywall to FreeS/WAN instructions \(PSK\)](#)

[Zyxel's Prestige to FreeS/WAN instructions \(PSK\)](#). Note: not all Prestige versions include VPN software.

[Fabrice Cahen's HOWTO \(PSK\)](#)

[Back to chart](#)

# Performance of FreeS/WAN

The performance of FreeS/WAN is adequate for most applications.

In normal operation, the main concern is the overhead for encryption, decryption and authentication of the actual IPsec (ESP and/or AH) data packets. Tunnel setup and rekeying occur so much less frequently than packet processing that, in general, their overheads are not worth worrying about.

At startup, however, tunnel setup overheads may be significant. If you reboot a gateway and it needs to establish many tunnels, expect some delay. This and other issues for large gateways are discussed below.

## Published material

The University of Wales at Aberystwyth has done quite detailed speed tests and put their results on the web.

Davide Cerri's thesis (in Italian) includes performance results for FreeS/WAN and for TLS. He posted an English summary on the mailing list.

Steve Bellovin used one of AT&T Research's FreeS/WAN gateways as his data source for an analysis of the cache sizes required for key swapping in IPsec. Available as text or PDF slides for a talk on the topic.

See also the NAI work mentioned in the next section.

## Estimating CPU overheads

We can come up with a formula that roughly relates CPU speed to the rate of IPsec processing possible. It is far from exact, but should be usable as a first approximation.

An analysis of authentication overheads for high-speed networks, including some tests using FreeS/WAN, is on the NAI Labs site. In particular, see figure 3 in this PDF document. Their estimates of overheads, measured in Pentium II cycles per byte processed are:

	IPsec	authentication	encryption	cycles/byte
Linux IP stack alone	no	no	no	5
IPsec without crypto	yes	no	no	11
IPsec, authentication only	yes	SHA-1	no	24
IPsec with encryption	yes	yes	yes	not tested

Overheads for IPsec with encryption were not tested in the NAI work, but Antoon Bosselaers' web page gives cost for his optimised Triple DES implementation as 928 Pentium cycles per block, or 116 per byte. Adding that to the 24 above, we get 140 cycles per byte for IPsec with encryption.

At 140 cycles per byte, a 140 MHz machine can handle a megabyte — 8 megabits — per second. Speeds for other machines will be proportional to this. To saturate a link with capacity C megabits per second, you need a machine running at  $C * 140/8 = C * 17.5$  MHz.

However, that estimate is not precise. It ignores the differences between:

## Introduction to FreeS/WAN

- NAI's test packets and real traffic
- NAI's Pentium II cycles, Bosselaers' Pentium cycles, and your machine's cycles
- different 3DES implementations
- SHA-1 and MD5

and does not account for some overheads you will almost certainly have:

- communication on the client-side interface
- switching between multiple tunnels — re-keying, cache reloading and so on

so we suggest using  $C * 25$  to get an estimate with a bit of a built-in safety factor.

This covers only IP and IPsec processing. If you have other loads on your gateway — for example if it is also working as a firewall — then you will need to add your own safety factor atop that.

This estimate matches empirical data reasonably well. For example, Metherringham's tests, described [below](#), show a 733 topping out between 32 and 36 Mbit/second, pushing data as fast as it can down a 100 Mbit link. Our formula suggests you need at least an 800 to handle a fully loaded 32 Mbit link. The two results are consistent.

Some examples using this estimation method:

Interface		Machine speed in MHz		
Type	Mbit per second	Estimate Mbit*25	Minimum IPSEC gateway	Minimum with other load (e.g. firewall)
DSL	1	25 MHz	whatever you have	133, or better if you have it
cable modem	3	75 MHz		
<i>any link, light load</i>	5	125 MHz	133	200+, <i>almost any surplus machine</i>
Ethernet	10	250 MHz	surplus 266 or 300	500+
<i>fast link, moderate load</i>	20	500 MHz	500	800+, <i>any current off-the-shelf PC</i>
T3 or E3	45	1125 MHz	1200	1500+
fast Ethernet	100	2500 MHz	// not feasible with 3DES in software on current machines //	
OC3	155	3875 MHz		

Such an estimate is far from exact, but should be usable as minimum requirement for planning. The key observations are:

- older *surplus machines* are fine for IPsec gateways at loads up to **5 megabits per second** or so
- a *mid-range new machine* can handle IPsec at rates up to **20 megabits per second** or more

## Higher performance alternatives

[AES](#) is a new US government block cipher standard, designed to replace the obsolete [DES](#). If FreeS/WAN using [3DES](#) is not fast enough for your application, the [AES patch](#) may help.

To date (March 2002) we have had only one [mailing list report](#) of measurements with the patch applied. It

indicates that, at least for the tested load on that user's network, *AES roughly doubles IPsec throughput*. If further testing confirms this, it may prove possible to saturate an OC3 link in software on a high-end box.

Also, some work is being done toward support of hardware IPsec acceleration which might extend the range of requirements FreeS/WAN could meet.

## Other considerations

CPU speed may be the main issue for IPsec performance, but of course it isn't the only one.

You need good ethernet cards or other network interface hardware to get the best performance. See this [ethernet information](#) page and this [Linux network driver](#) page.

The current FreeS/WAN kernel code is largely single-threaded. It is SMP safe, and will run just fine on a multiprocessor machine ([discussion](#)), but the load within the kernel is not shared effectively. This means that, for example to saturate a T3 — which needs about a 1200 MHz machine — you cannot expect something like a dual 800 to do the job.

On the other hand, SMP machines do tend to share loads well so — provided one CPU is fast enough for the IPsec work — a multiprocessor machine may be ideal for a gateway with a mixed load.

## Many tunnels from a single gateway

FreeS/WAN allows a single gateway machine to build tunnels to many others. There may, however, be some problems for large numbers as indicated in this message from the mailing list:

```
Subject: Re: Maximum number of ipsec tunnels?
Date: Tue, 18 Apr 2000
From: "John S. Denker" <jsd@research.att.com>
```

Christopher Ferris wrote:

```
>> What are the maximum number ipsec tunnels FreeS/WAN can handle??
```

Henry Spencer wrote:

```
>There is no particular limit. Some of the setup procedures currently
>scale poorly to large numbers of connections, but there are (clumsy)
>workarounds for that now, and proper fixes are coming.
```

```
1) "Large" numbers means anything over 50 or so. I routinely run boxes
with about 200 tunnels. Once you get more than 50 or so, you need to worry
about several scalability issues:
```

```
a) You need to put a "-" sign in syslogd.conf, and rotate the logs daily
not weekly.
```

```
b) Processor load per tunnel is small unless the tunnel is not up, in which
case a new half-key gets generated every 90 seconds, which can add up if
you've got a lot of down tunnels.
```

```
c) There's other bits of lore you need when running a large number of
tunnels. For instance, systematically keeping the .conf file free of
conflicts requires tools that aren't shipped with the standard freeswan
package.
```

## Introduction to FreeS/WAN

d) The pluto startup behavior is quadratic. With 200 tunnels, this eats up several minutes at every restart. I'm told fixes are coming soon.

2) Other than item (1b), the CPU load depends mainly on the size of the pipe attached, not on the number of tunnels.

It is worth noting that item (1b) applies only to repeated attempts to re-key a data connection (IPsec SA, Phase 2) over an established keying connection (ISAKMP SA, Phase 1). There are two ways to reduce this overhead using settings in [ipsec.conf\(5\)](#):

- set *keyingtries* to some small value to limit repetitions
- set *keylife* to a short time so that a failing data connection will be cleaned up when the keying connection is reset.

The overheads for establishing keying connections (ISAKMP SAs, Phase 1) are lower because for these Pluto does not perform expensive operations before receiving a reply from the peer.

A gateway that does a lot of rekeying — many tunnels and/or low settings for tunnel lifetimes — will also need a lot of [random numbers](#) from the random(4) driver.

## Low-end systems

*Even a 486 can handle a T1 line, according to this mailing list message:*

```
Subject: Re: linux-ipsec: IPSec Masquerade
Date: Fri, 15 Jan 1999 11:13:22 -0500
From: Michael Richardson
```

```
. . . A 486/66 has been clocked by Phil Karn to do
10Mb/s encryption.. that uses all the CPU, so half that to get some CPU,
and you have 5Mb/s. 1/3 that for 3DES and you get 1.6Mb/s....
```

and a piece of mail from project technical lead Henry Spencer:

```
Oh yes, and a new timing point for Sandy's docs... A P60 -- yes, a 60MHz
Pentium, talk about antiques -- running a host-to-host tunnel to another
machine shows an FTP throughput (that is, end-to-end results with a real
protocol) of slightly over 5Mbit/s either way. (The other machine is much
faster, the network is 100Mbps, and the ether cards are good ones... so
the P60 is pretty definitely the bottleneck.)
```

From the above, and from general user experience as reported on the list, it seems clear that a cheap surplus machine — a reasonable 486, a minimal Pentium box, a Sparc 5, ... — can easily handle a home office or a small company connection using any of:

- ADSL service
- cable modem
- T1
- E1

If available, we suggest using a Pentium 133 or better. This should ensure that, even under maximum load, IPsec will use less than half the CPU cycles. You then have enough left for other things you may want on your gateway — firewalling, web caching, DNS and such.

## Measuring KLIPS

Here is some additional data from the mailing list.

Subject: FreeSWAN (specically KLIPS) performance measurements  
 Date: Thu, 01 Feb 2001  
 From: Nigel Metheringham <Nigel.Metheringham@intechology.co.uk>

I've spent a happy morning attempting performance tests against KLIPS (this is due to me not being able to work out the CPU usage of KLIPS so resorting to the crude measurements of maximum throughput to give a baseline to work out loading of a box).

Measurements were done using a set of 4 boxes arranged in a line, each connected to the next by 100Mbit duplex ethernet. The inner 2 had an ipsec tunnel between them (shared secret, but I was doing measurements when the tunnel was up and running - keying should not be an issue here). The outer pair of boxes were traffic generators or traffic sink.

The crypt boxes are Compaq DL380s - Uniprocessor PIII/733 with 256K cache. They have 128M main memory. Nothing significant was running on the boxes other than freeswan. The kernel was a 2.2.19pre7 patched with freeswan and ext3.

Without an ipsec tunnel in the chain (ie the 2 inner boxes just being 100BaseT routers), throughput (measured with ttcp) was between 10644 and 11320 KB/sec

With an ipsec tunnel in place, throughput was between 3268 and 3402 KB/sec

These measurements are for data pushed across a TCP link, so the traffic on the wire between the 2 ipsec boxes would have been higher than this....

vmstat (run during some other tests, so not affecting those figures) on the encrypting box shows approx 50% system & 50% idle CPU - which I don't believe at all. Interactive feel of the box was significantly sluggish.

I also tried running the kernel profiler (see man readprofile) during test runs.

A box doing primarily decrypt work showed basically nothing happening - I assume interrupts were off.

A box doing encrypt work showed the following:-

Ticks	Function	Load
956	total	0.0010
532	des_encrypt2	0.1330
110	MD5Transform	0.0443
97	kmalloc	0.1880
39	des_encrypt3	0.1336
23	speedo_interrupt	0.0298
14	skb_copy_expand	0.0250
13	ipsec_tunnel_start_xmit	0.0009
13	Decode	0.1625
11	handle_IRQ_event	0.1019
11	.des_ncbc_encrypt_end	0.0229
10	speedo_start_xmit	0.0188
9	satoa	0.0225

## Introduction to FreeS/WAN

8	kfree	0.0118
8	ip_fragment	0.0121
7	ultoa	0.0365
5	speedo_rx	0.0071
5	.des_encrypt2_end	5.0000
4	_stext	0.0140
4	ip_fw_check	0.0035
2	rj_match	0.0034
2	ipfw_output_check	0.0200
2	inet_addr_type	0.0156
2	eth_copy_and_sum	0.0139
2	dev_get	0.0294
2	addrtoa	0.0143
1	speedo_tx_buffer_gc	0.0024
1	speedo_refill_rx_buf	0.0022
1	restore_all	0.0667
1	number	0.0020
1	net_bh	0.0021
1	neigh_connected_output	0.0076
1	MD5Final	0.0083
1	kmem_cache_free	0.0016
1	kmem_cache_alloc	0.0022
1	__kfree_skb	0.0060
1	ipsec_rcv	0.0001
1	ip_rcv	0.0014
1	ip_options_fragment	0.0071
1	ip_local_deliver	0.0023
1	ipfw_forward_check	0.0139
1	ip_forward	0.0011
1	eth_header	0.0040
1	.des_encrypt3_end	0.0833
1	des_decrypt3	0.0034
1	csum_partial_copy_generic	0.0045
1	call_out_firewall	0.0125

Hope this data is helpful to someone... however the lack of visibility into the decrypt side makes things less clear

## Speed with compression

Another user reported some results for connections with and without IP compression:

Subject: [Users] Speed with compression  
Date: Fri, 29 Jun 2001  
From: John McMonagle <johnm@advocap.org>

Did a couple tests with compression using the new 1.91 freeswan.

Running between 2 sites with cable modems. Both using approximately 130 mhz pentium.

Transferred files with ncftp.

Compressed file was a 6mb compressed installation file.  
Non compressed was 18mb /var/lib/rpm/packages.rpm

	Compressed vpn	regular vpn
Compress file	42.59 kBs	42.08 kBs
regular file	110.84 kBs	41.66 kBs

## Introduction to FreeS/WAN

Load was about 0 either way.  
Ping times were very similar a bit above 9 ms.

Compression looks attractive to me.

Later in the same thread, project technical lead Henry Spencer added:

```
> is there a reason not to switch compression on? I have large gateway boxes  
> connecting 3 connections, one of them with a measly DS1 link...
```

Run some timing tests with and without, with data and loads representative of what you expect in production. That's the definitive way to decide. If compression is a net loss, then obviously, leave it turned off. If it doesn't make much difference, leave it off for simplicity and hence robustness. If there's a substantial gain, by all means turn it on.

If both ends support compression and can successfully negotiate a compressed connection (trivially true if both are FreeS/WAN 1.91), then the crucial question is CPU cycles.

Compression has some overhead, so one question is whether *your* data compresses well enough to save you more CPU cycles (by reducing the volume of data going through CPU-intensive encryption/decryption) than it costs you. Last time I ran such tests on data that was reasonably compressible but not deliberately contrived to be so, this generally was not true -- compression cost extra CPU cycles -- so compression was worthwhile only if the link, not the CPU, was the bottleneck. However, that was before the slow-compression bug was fixed. I haven't had a chance to re-run those tests yet, but it sounds like I'd probably see a different result.

The bug he refers to was a problem with the compression libraries that had us using C code, rather than assembler, for compression. It was fixed before 1.91.

## Methods of measuring

If you want to measure the loads FreeS/WAN puts on a system, note that tools such as top or measurements such as load average are more-or-less useless for this. They are not designed to measure something that does most of its work inside the kernel.

Here is a message from FreeS/WAN kernel programmer Richard Guy Briggs on this:

```
> I have a batch of boxes doing Freeswan stuff.  
> I want to measure the CPU loading of the Freeswan tunnels, but am  
> having trouble seeing how I get some figures out...  
>  
> - Keying etc is in userspace so will show up on the per-process  
>   and load average etc (ie pluto's load)
```

Correct.

```
> - KLIPS is in the kernel space, and does not show up in load average  
>   I think also that the KLIPS per-packet processing stuff is running  
>   as part of an interrupt handler so it does not show up in the  
>   /proc/stat system_cpu or even idle_cpu figures
```

It is not running in interrupt handler. It is in the bottom half. This is somewhere between user context (careful, this is not userspace!) and hardware interrupt context.

## Introduction to FreeS/WAN

```
> Is this correct, and is there any means of instrumenting how much the  
> cpu is being loaded - I don't like the idea of a system running out of  
> steam whilst still showing 100% idle CPU :-)
```

vmstat seems to do a fairly good job, but use a running tally to get a good idea. A one-off call to vmstat gives different numbers than a running stat. To do this, put an interval on your vmstat command line.

and another suggestion from the same thread:

```
Subject: Re: Measuring the CPU usage of Freeswan  
Date: Mon, 29 Jan 2001  
From: Patrick Michael Kane <modus@pr.es.to>
```

The only truly accurate way to accurately track FreeSWAN CPU usage is to use a CPU soaker. You run it on an unloaded system as a benchmark, then start up FreeSWAN and take the difference to determine how much FreeSWAN is eating. I believe someone has done this in the past, so you may find something in the FreeSWAN archives. If not, someone recently posted a URL to a CPU soaker benchmark tool on linux-kernel.

# Testing FreeS/WAN

This document discusses testing FreeS/WAN.

Not all types of testing are described here. Other parts of the documentation describe some tests:

[installation document](#)

testing for a successful install

[configuration document](#)

basic tests for a working configuration

[web links document](#)

General information on tests for interoperability between various IPsec implementations. This includes links to several test sites.

[interoperation document](#).

More specific information on FreeS/WAN interoperation with other implementations.

[performance document](#)

performance measurements

The test setups and procedures described here can also be used in other testing, but this document focuses on testing the IPsec functionality of FreeS/WAN.

## Testing opportunistic connections

This section teaches you how to test your opportunistically encrypted (OE) connections. To set up OE, please see the easy instructions in our [quickstart guide](#).

### Basic OE Test

This test is for basic OE functionality. For additional tests, keep reading.

Be sure IPsec is running. You can see whether it is with:

```
ipsec setup status
```

If need be, you can restart it with:

```
service ipsec restart
```

Load a FreeS/WAN test website from the host on which you're running FreeS/WAN. Note: the feds may be watching these sites. Type one of:

```
links oetest.freeswan.org
```

```
links oetest.freeswan.nl
```

A positive result looks like this:

```
You seem to be connecting from: 192.0.2.11 which DNS says is:
gateway.example.com
```

---

## Introduction to FreeS/WAN

```
Status E-route
OE      enabled    16    192.139.46.73/32    ->    192.0.2.11/32    =>
tun0x2097@192.0.2.11
OE      enabled    176   192.139.46.77/32    ->    192.0.2.11/32    =>
tun0x208a@192.0.2.11
```

If you see this, congratulations! Your OE box will now encrypt its own traffic whenever it can. If you have difficulty, see our [OE troubleshooting tips](#).

## OE Gateway Test

If you've set up FreeS/WAN to protect a subnet behind your gateway, you'll need to run another simple test, which can be done from a machine running any OS. That's right, your Windows box can be protected by opportunistic encryption without any FreeS/WAN install or configuration on that box. From *each protected subnet node*, load the FreeS/WAN website with:

```
links oetest.freeswan.org
```

```
links oetest.freeswan.nl
```

A positive result looks like this:

```
You seem to be connecting from: 192.0.2.98 which DNS says is:
box98.example.com
```

---

```
Status E-route
OE      enabled    16    192.139.46.73/32    ->    192.0.2.98/32    =>
tun0x134ed@192.0.2.11
OE      enabled    176   192.139.46.77/32    ->    192.0.2.11/32    =>
tun0x134d2@192.0.2.11
```

If you see this, congratulations! Your OE gateway will now encrypt traffic for this subnet node whenever it can. If you have difficulty, see our [OE troubleshooting tips](#).

## Additional OE tests

When testing OE, you will often find it useful to execute this command on the FreeS/WAN host:

```
ipsec eroute
```

If you have established a connection (either for or for a subnet node) you will see a result like:

```
192.0.2.11/32    -> 192.139.46.73/32    => tun0x149f@192.139.46.38
```

Key:

1. 192.0.2.11/32 Local start point of the protected traffic.
2. 192.0.2.194/32 Remote end point of the protected traffic.
3. 192.0.48.38 Remote FreeS/WAN node (gateway or host). May be the same as (2).
4. [not shown] Local FreeS/WAN node (gateway or host), where you've produced the output. May be the same as (1).

For extra assurance, you may wish to use a packet sniffer such as [tcpdump](#) to verify that packets are being encrypted. You should see output that indicates *ESP* encrypted data, for example:

```
02:17:47.353750 PPPoE [ses 0x1e12] IP 154: xy.example.com > oetest.freeswan.org: ESP spi=0x8
```

## Testing with User Mode Linux

[User Mode Linux](#) allows you to run Linux as a user process on another Linux machine.

As of 1.92, the distribution has a new directory named testing. It contains a collection of test scripts and sample configurations. Using these, you can bring up several copies of Linux in user mode and have them build tunnels to each other. This lets you do some testing of a FreeS/WAN configuration on a single machine.

You need a moderately well-endowed machine for this to work well. Each UML wants about 16 megs of memory by default, which is plenty for FreeS/WAN usage. Typical regression testing only occasionally uses as many as 4 UMLs. If one is doing nothing else with the machine (in particular, not running X on it), then 128 megs and a 500MHz CPU are fine.

Documentation on these scripts is [here](#). There is also documentation on automated testing [here](#).

## Configuration for a testbed network

A common test setup is to put a machine with dual Ethernet cards in between two gateways under test. You need at least five machines; two gateways, two clients and a testing machine in the middle.

The central machine both routes packets and provides a place to run diagnostic software for checking IPsec packets. See next section for discussion of [using tcpdump\(8\)](#) for this.

This makes things more complicated than if you just connected the two gateway machines directly to each other, but it also makes your test setup much more like the environment you actually use IPsec in. Those environments nearly always involve routing, and quite a few apparent IPsec failures turn out to be problems with routing or with firewalls dropping packets. This approach lets you deal with those problems on your test setup.

What you end up with looks like:

### Testbed network

```
subnet a.b.c.0/24
|
eth1 = a.b.c.1
  gate1
eth0 = 192.168.p.1
|
eth0 = 192.168.p.2
  route/monitor box
eth1 = 192.168.q.2
|
eth0 = 192.168.q.1
  gate2
```

## Introduction to FreeS/WAN

```
eth1 = x.y.z.1
      |
      subnet x.y.z.0/24
```

Where p and q are any convenient values that do not interfere with other routes you may have. The ipsec.conf(5) file then has, among other things:

```
conn abc-xyz
    left=192.168.p.1
    leftnexthop=192.168.p.2
    right=192.168.q.1
    rightnexthop=192.168.q.2
```

Once that works, you can remove the "route/monitor box", and connect the two gateways to the Internet. The only parameters in ipsec.conf(5) that need to change are the four shown above. You replace them with values appropriate for your Internet connection, and change the eth0 IP addresses and the default routes on both gateways.

Note that nothing on either subnet needs to change. This lets you test most of your IPsec setup before connecting to the insecure Internet.

## Using packet sniffers in testing

A number of tools are available for looking at packets. We will discuss using [tcpdump\(8\)](#), a common Linux tool included in most distributions. Alternatives offering more-or-less the same functionality include:

### Ethereal

Several people on our mailing list report a preference for this over tcpdump.

### windump

a Windows version of tcpdump(8), possibly handy if you have Windows boxes in your network

### Sniffit

A linux sniffer that we don't know much about. If you use it, please comment on our mailing list.

See also this [index](#) of packet sniffers.

tcpdump(8) may misbehave if run on the gateways themselves. It is designed to look into a normal IP stack and may become confused if you ask it to display data from a stack which has IPsec in play.

At one point, the problem was quite severe. Recent versions of tcpdump, however, understand IPsec well enough to be usable on a gateway. You can get the latest version from [tcpdump.org](#).

Even with a recent tcpdump, some care is required. Here is part of a post from Henry on the topic:

```
> a) data from sunset to sunrise or the other way is not being
> encrypted (I am using tcpdump (ver. 3.4) -x/ping -p to check
> packages)
```

What *\*interface\** is tcpdump being applied to? Use the -i option to control this. It matters! If tcpdump is looking at the ipsecN interfaces, e.g. ipsec0, then it is seeing the packets before they are encrypted or after they are decrypted, so of course they don't look encrypted. You want to have tcpdump looking at the actual hardware interfaces, e.g. eth0.

Actually, the only way to be *\*sure\** what you are sending on the wire is to

## Introduction to FreeS/WAN

have a separate machine eavesdropping on the traffic. Nothing you can do on the machines actually running IPsec is 100% guaranteed reliable in this area (although tcpdump is a lot better now than it used to be).

The most certain way to examine IPsec packets is to look at them on the wire. For security, you need to be certain, so we recommend doing that. To do so, you need a *separate sniffer machine located between the two gateways*. This machine can be routing IPsec packets, but it must not be an IPsec gateway. Network configuration for such testing is discussed above.

Here's another mailing list message with advice on using tcpdump(8):

Subject: RE: [Users] Encrypted???

Date: Thu, 29 Nov 2001

From: "Joe Patterson" <jpatterson@asgardgroup.com>

```
tcpdump -nl -i $EXT-IF proto 50
```

-nl tells it not to buffer output or resolve names (if you don't do that it may confuse you by not outputting anything for a while), -i \$EXT-IF (replace with your external interface) tells it what interface to listen on, and proto 50 is ESP. Use "proto 51" if for some odd reason you're using AH, and "udp port 500" if you want to see the isakmp key exchange/tunnel setup packets.

You can also run `tcpdump -nl -i ipsec0` to see what traffic is on that virtual interface. Anything you see there *should* be either encrypted or dropped (unless you've turned on some strange options in your ipsec.conf file)

Another very handy thing is ethereal (<http://www.ethereal.com/>) which runs on just about anything, has a nice gui interface (or a nice text-based interface), and does a great job of protocol breakdown. For ESP and AH it'll basically just tell you that there's a packet of that protocol, and what the spi is, but for isakmp it'll actually show you a lot of the tunnel setup information (until it gets to the point in the protocol where isakmp is encrypted....)

## Verifying encryption

The question of how to verify that messages are actually encrypted has been extensively discussed on the mailing list. See this [thread](#).

If you just want to verify that packets are encrypted, look at them with a packet sniffer (see [previous section](#)) located between the gateways. The packets should, except for some of the header information, be utterly unintelligible. *The output of good encryption looks exactly like random noise.*

A packet sniffer can only tell you that the data you looked at was encrypted. If you have stronger requirements — for example if your security policy requires verification that plaintext is not leaked during startup or under various anomolous conditions — then you will need to devise much more thorough tests. If you do that, please post any results or methodological details which your security policy allows you to make public.

You can put recognizable data into ping packets with something like:

```
ping -p feedfacedeadbeef 11.0.1.1
```

"feedfacedeadbeef" is a legal hexadecimal pattern that is easy to pick out of hex dumps.

For other protocols, you may need to check if you have encrypted data or ASCII text. Encrypted data has approximately equal frequencies for all 256 possible characters. ASCII text has most characters in the printable range 0x20–0x7f, a few control characters less than 0x20, and none at all in the range 0x80–0xff. 0x20, space, is a good character to look for. In normal English text space occurs about once in seven characters, versus about once in 256 for random or encrypted data.

One thing to watch for: the output of good compression, like that of good encryption, looks just like random noise. You cannot tell just by looking at a data stream whether it has been compressed, encrypted, or both. You need a little care not to mistake compressed data for encrypted data in your testing.

Note also that weak encryption also produces random-looking output. You cannot tell whether the encryption is strong by looking at the output. To be sure of that, you would need to have both the algorithms and the implementation examined by experts.

For IPsec, you can get partial assurance from interoperability tests. See our [interop](#) document. When twenty products all claim to implement 3DES, and they all talk to each other, you can be fairly sure they have it right. Of course, you might wonder whether all the implementers are conspiring to trick you or, more plausibly, whether some implementations might have "back doors" so they can get also it wrong when required.. If you're seriously worried about things like that, you need to get the code you use audited (good luck if it is not Open Source), or perhaps to talk to a psychiatrist about treatments for paranoia.

## Mailing list pointers

Additional information on testing can be found in these [mailing list](#) messages:

- a user's detailed [setup diary](#) for his testbed network
- a FreeS/WAN team member's [notes](#) from testing at an IPsec interop "bakeoff"

# Kernel configuration for FreeS/WAN

This section lists many of the options available when configuring a Linux kernel, and explains how they should be set on a FreeS/WAN IPsec gateway.

## Not everyone needs to worry about kernel configuration

Note that in many cases you do not need to mess with these.

You may have a Linux distribution which comes with FreeS/WAN installed (see this [list](#)). In that case, you need not do a FreeS/WAN installation or a kernel configuration. Of course, you might still want to configure and rebuild your kernel to improve performance or security. This can be done with standard tools described in the [Kernel HowTo](#).

If you need to install FreeS/WAN, then you do need to configure a kernel. However, you may choose to do that using the simplest procedure:

- Configure, build and test a kernel for your system before adding FreeS/WAN. See the [Kernel HowTo](#) for details. ***This step cannot be skipped.*** FreeS/WAN needs the results of your configuration.
- Then use FreeS/WAN's *make oldgo* command. This sets everything FreeS/WAN needs and retains your values everywhere else.

This document is for those who choose to configure their FreeS/WAN kernel themselves.

## Assumptions and notation

Help text for most kernel options is included with the kernel files, and is accessible from within the configuration utilities. We assume you will refer to that, and to the [Kernel HowTo](#), as necessary. This document covers only the FreeS/WAN-specific aspects of the problem.

To avoid duplication, this document section does not cover settings for the additional IPsec-related kernel options which become available after you have patched your kernel with FreeS/WAN patches. There is help text for those available from within the configuration utility.

We assume a common configuration in which the FreeS/WAN IPsec gateway is also doing ipchains(8) firewalling for a local network, and possibly masquerading as well.

Some suggestions below are labelled as appropriate for "a true paranoid". By this we mean they may cause inconvenience and it is not entirely clear they are necessary, but they appear to be the safest choice. Not using them might entail some risk. Of course one suggested mantra for security administrators is: "I know I'm paranoid. I wonder if I'm paranoid enough."

## Labels used

Six labels are used to indicate how options should be set. We mark the labels with [square brackets]. For two of these labels, you have no choice:

*[required]*

essential for FreeS/WAN operation.

*[incompatible]*

incompatible with FreeS/WAN.

those must be set correctly or FreeS/WAN will not work

FreeS/WAN should work with any settings of the others, though of course not all combinations have been tested. We do label these in various ways, but *these labels are only suggestions*.

*[recommended]*

useful on most FreeS/WAN gateways

*[disable]*

an unwelcome complication on a FreeS/WAN gateway.

*[optional]*

Your choice. We outline issues you might consider.

*[anything]*

This option has no direct effect on FreeS/WAN and related tools, so you should be able to set it as you please.

Of course complexity is an enemy in any effort to build secure systems. ***For maximum security, any feature that can reasonably be turned off should be.*** "If in doubt, leave it out."

## Kernel options for FreeS/WAN

Indentation is based on the nesting shown by 'make menuconfig' with a 2.2.16 kernel for the i386 architecture.

*Code maturity and level options*

*Prompt for development ... code/drivers*

[optional] If this is ***no***, experimental drivers are not shown in later menus.

For most FreeS/WAN work, ***no*** is the preferred setting. Using new or untested components is too risky for a security gateway.

However, for some hardware (such as the author's network cards) the only drivers available are marked ***new/experimental***. In such cases, you must enable this option or your cards will not appear under "network device support". A true paranoid would leave this option off and replace the cards.

*Processor type and features*

[anything]

*Loadable module support*

*Enable loadable module support*

[optional] A true paranoid would disable this. An attacker who has root access to your machine can fairly easily install a bogus module that does awful things, provided modules are enabled. A common tool for attackers is a "rootkit", a set of tools the attacker uses once he or she has become root on your system. The kit introduces assorted additional compromises so that the attacker will continue to "own" your system despite most things you might do to recovery the situation. For Linux, there is a tool called knark which is basically a rootkit packaged as a kernel module.

## Introduction to FreeS/WAN

With modules disabled, an attacker cannot install a bogus module. The only way he can achieve the same effects is to install a new kernel and reboot. This is considerably more likely to be noticed.

Many FreeS/WAN gateways run with modules enabled. This simplifies some administrative tasks and some ipchains features are available only as modules. Once an enemy has root on your machine your security is nil, so arguably defenses which come into play only in that situation are pointless.

*Set version information ....*

[optional] This provides a check to prevent loading modules compiled for a different kernel.

*Kernel module loader*

[disable] It gives little benefit on a typical FreeS/WAN gate and entails some risk.

*General setup*

We list here only the options that matter for FreeS/WAN.

*Networking support*

[required]

*Sysctl interface*

[optional] If this option is turned on and the */proc* filesystem installed, then you can control various system behaviours by writing to files under */proc/sys*. For example:

```
echo 1 > /proc/sys/net/ipv4/ipforward
```

turns IP forwarding on.

Disabling this option breaks many firewall scripts. A true paranoid would disable it anyway since it might conceivably be of use to an attacker.

*Plug and Play support*

[anything]

*Block devices*

[anything]

*Networking options*

*Packet socket*

[optional] This kernel feature supports tools such as tcpdump(8) which communicate directly with network hardware, bypassing kernel protocols. This is very much a two-edged sword:

◊ such tools can be very useful to the firewall admin, especially during initial testing

◊ should an evildoer breach your firewall, such tools could give him or her a great deal of information about the rest of your network

We recommend disabling this option on production gateways.

*Kernel/User netlink socket*

[optional] Required if you want to use advanced router features.

*Routing messages*

[optional]

*Netlink device emulation*

[optional]

*Network firewalls*

[recommended] You need this if the IPsec gateway also functions as a firewall.

## Introduction to FreeS/WAN

Even if the IPsec gateway is not your primary firewall, we suggest setting this so that you can protect the gateway with at least basic local packet filters.

### *Socket filtering*

[disable] This enables an older filtering interface. We suggest using `ipchains(8)` instead. To do that, set the "Network firewalls" option just above, and not this one.

### *Unix domain sockets*

[required] These sockets are used for communication between the `ipsec(8)` commands and the `ipsec pluto(8)` daemon.

### *TCP/IP networking*

[required]

#### *IP: multicasting*

[anything]

#### *IP: advanced router*

[optional] This gives you policy routing, which some people have used to good advantage in their scripts for FreeS/WAN gateway management. It is not used in our distributed scripts, so not required unless you want it for custom scripts. It requires the `netlink` interface between kernel code and the `iproute2(8)` command.

#### *IP: kernel level autoconfiguration*

[disable] It gives little benefit on a typical FreeS/WAN gate and entails some risk.

#### *IP: firewall packet netlink device*

[disable]

#### *IP: transparent proxy support*

[optional] This is required in some firewall configurations, but should be disabled unless you have a definite need for it.

#### *IP: masquerading*

[optional] Required if you want to use `non-routable` private IP addresses for your local network.

#### *IP: Optimize as router not host*

[recommended]

#### *IP: tunneling*

[required]

#### *IP: GRE tunnels over IP*

[anything]

#### *IP: aliasing support*

[anything]

#### *IP: ARP daemon support (EXPERIMENTAL)*

Not required on most systems, but might prove useful on heavily-loaded gateways.

#### *IP: TCP syncookie support*

[recommended] It provides a defense against a `denial of service attack` which uses bogus TCP connection requests to waste resources on the victim machine.

#### *IP: Reverse ARP*

#### *IP: large window support*

[recommended] unless you have less than 16 meg RAM

### *IPv6*

[optional] FreeS/WAN does not currently support IPv6, though work on integrating FreeS/WAN with the Linux IPv6 stack has begun. [Details](#).

## Introduction to FreeS/WAN

It should be possible to use IPv4 FreeS/WAN on a machine which also does IPv6. This combination is not yet well tested. We would be quite interested in hearing results from anyone experimenting with it, via the [mailing list](#).

We do not recommend using IPv6 on production FreeS/WAN gateways until more testing has been done.

### *Novell IPX*

[disable]

### *Appletalk*

[disable] Quite a few Linux installations use IP but also have some other protocol, such as Appletalk or IPX, for communication with local desktop machines. In theory it should be possible to configure IPsec for the IP side of things without interfering with the second protocol.

We do not recommend this. Keep the software on your gateway as simple as possible. If you need a Linux-based Appletalk or IPX server, use a separate machine.

### *Telephony support*

[anything]

### *SCSI support*

[anything]

### *I2O device support*

[anything]

### *Network device support*

[anything] should work, but there are some points to note.

The development team test almost entirely on 10 or 100 megabit Ethernet and modems. In principle, any device that can do IP should be just fine for IPsec, but in the real world any device that has not been well-tested is somewhat risky. By all means try it, but don't bet your project on it until you have solid test results.

If you disabled experimental drivers in the [Code maturity](#) section above, then those drivers will not be shown here. Check that option before going off to hunt for missing drivers.

If you want Linux to automatically find more than one ethernet interface at boot time, you need to:

- compile the appropriate driver(s) into your kernel. Modules will not work for this
- add a line such as

```
append="ether=0,0,eth0 ether=0,0,eth1"
```

to your /etc/lilo.conf file. In some cases you may need to specify parameters such as IRQ or base address. The example uses "0,0" for these, which tells the system to search. If the search does not succeed on your hardware, then you should retry with explicit parameters. See the lilo.conf(5) man page for details.

- run lilo(8)

Having Linux find the cards this way is not necessary, but is usually more convenient than loading modules in your boot scripts.

### *Amateur radio support*

[anything]

### *IrDA (infrared) support*

[anything]

*ISDN subsystem*

[anything]

*Old CDROM drivers*

[anything]

*Character devices*

The only required character device is:

*random(4)*

[required] This is a source of random numbers which are required for many cryptographic protocols, including several used in IPsec.

If you are comfortable with C source code, it is likely a good idea to go in and adjust the *#define* lines in */usr/src/linux/drivers/char/random.c* to ensure that all sources of randomness are enabled. Relying solely on keyboard and mouse randomness is dubious procedure for a gateway machine. You could also increase the randomness pool size from the default 512 bytes (128 32-bit words).

*Filesystems*

[anything] should work, but we suggest limiting a gateway machine to the standard Linux ext2 filesystem in most cases.

*Network filesystems*

[disable] These systems are an unnecessary risk on an IPsec gateway.

*Console drivers*

[anything]

*Sound*

[anything] should work, but we suggest enabling sound only if you plan to use audible alarms for firewall problems.

*Kernel hacking*

[disable] This might be enabled on test machines, but should not be on production gateways.

# Other configuration possibilities

This document describes various options for FreeS/WAN configuration which are less used or more complex (often both) than the standard cases described in our [config](#) and [quickstart](#) documents.

## Some rules of thumb about configuration

### Tunnels are cheap

Nearly all of the overhead in IPsec processing is in the encryption and authentication of packets. Our [performance](#) document discusses these overheads.

Beside those overheads, the cost of managing additional tunnels is trivial. Whether your gateway supports one tunnel or ten just does not matter. A hundred might be a problem; there is a [section](#) on this in the performance document.

So, in nearly all cases, if using multiple tunnels gives you a reasonable way to describe what you need to do, you should describe it that way in your configuration files.

For example, one user recently asked on a mailing list about this network configuration:

```
netA---gwA---gwB---netB
                        |----netC

netA and B are secured netC not.
netA and gwA can not access netC
```

The user had constructed only one tunnel, netA to netB, and wanted to know how to use ip-route to get netC packets into it. This is entirely unnecessary. One of the replies was:

```
The simplest way and indeed the right way to
solve this problem is to set up two connections:
```

```
leftsubnet=NetA
left=gwA
right=gwB
rightsubnet=NetB
and
leftsubnet=NetA
left=gwA
right=gwB
rightsubnet=NetC
```

This would still be correct even if we added nets D, E, F, ... to the above diagram and needed twenty tunnels.

Of course another possibility would be to just use one tunnel, with a subnet mask that includes both netB and netC (or B, C, D, ...). See next section.

In general, you can construct as many tunnels as you need. Networks like netC in this example that do not connect directly to the gateway are fine, as long as the gateway can route to them.

The number of tunnels can become an issue if it reaches 50 or so. This is discussed in the [performance](#) document. Look there for information on supporting hundreds of Road Warriors from one gateway.

If you find yourself with too many tunnels for some reason like having eight subnets at one location and nine at another so you end up with  $9 \times 8 = 72$  tunnels, read the next section here.

### Subnet sizes

The subnets used in *leftsubnet* and *rightsubnet* can be of any size that fits your needs, and they need not correspond to physical networks.

You adjust the size by changing the subnet mask, the number after the slash in the subnet description. For example

- in 192.168.100.0/24 the /24 mask says 24 bits are used to designate the network. This leave 8 bits to label machines. This subnet has 256 addresses. .0 and .255 are reserved, so it can have 254 machines.
- A subnet with a /23 mask would be twice as large, 512 addresses.
- A subnet with a /25 mask would be half the size, 128 addresses.
- /0 is the whole Internet
- /32 is a single machine

As an example of using these in connection descriptions, suppose your company's head office has four physical networks using the address ranges:

```
192.168.100.0/24
    development
192.168.101.0/24
    production
192.168.102.0/24
    marketing
192.168.103.0/24
    administration
```

You can use exactly those subnets in your connection descriptions, or use larger subnets to grant broad access if required:

```
leftsubnet=192.168.100.0/24
    remote hosts can access only development
leftsubnet=192.168.100.0/23
    remote hosts can access development or production
leftsubnet=192.168.102.0/23
    remote hosts can access marketing or administration
leftsubnet=192.168.100.0/22
    remote hosts can access any of the four departments
```

or use smaller subnets to restrict access:

```
leftsubnet=192.168.103.0/24
    remote hosts can access any machine in administration
leftsubnet=192.168.103.64/28
```

## Introduction to FreeS/WAN

remote hosts can access only certain machines in administration.

*leftsubnet=192.168.103.42/32*

remote hosts can access only one particular machine in administration

To be exact, 192.68.103.64/28 means all addresses whose top 28 bits match 192.168.103.64. There are 16 of these because there are 16 possibilities for the remaining 4 bits. Their addresses are 192.168.103.64 to 192.168.103.79.

Each connection description can use a different subnet if required.

It is possible to use all the examples above on the same FreeS/WAN gateway, each in a different connection description, perhaps for different classes of user or for different remote offices.

It is also possible to have multiple tunnels using different *leftsubnet* descriptions with the same *right*. For example, when the marketing manager is on the road he or she might have access to:

*leftsubnet=192.168.102.0/24*

all machines in marketing

*192.168.101.32/29*

some machines in production

*leftsubnet=192.168.103.42/32*

one particular machine in administration

This takes three tunnels, but tunnels are cheap. If the laptop is set up to build all three tunnels automatically, then he or she can access all these machines concurrently, perhaps from different windows.

## Other network layouts

Here is the usual network picture for a site-to-site VPN::

```
Sunset=====West-----East=====Sunrise
      local net          untrusted net          local net
```

and for the Road Warrior::

```

                                     telecommuter's PC or
                                     traveller's laptop
Sunset=====West-----East
      corporate LAN          untrusted net
```

Other configurations are also possible.

## The Internet as a big subnet

A telecommuter might have:

```
Sunset=====West-----East ===== firewall --- the Internet
      home network          untrusted net          corporate network
```

This can be described as a special case of the general subnet-to-subnet connection. The subnet on the right is 0.0.0.0/0, the whole Internet.

West (the home gateway) can have its firewall rules set up so that only IPsec packets to East are allowed out. It will then behave as if its only connection to the world was a wire to East.

When machines on the home network need to reach the Internet, they do so via the tunnel, East and the corporate firewall. From the viewpoint of the Internet (perhaps of some EvilDoer trying to break in!), those home office machines are behind the firewall and protected by it.

### Wireless

Another possible configuration comes up when you do not trust the local network, either because you have very high security standards or because you are using easily-intercepted wireless signals.

Some wireless networks have built-in encryption called WEP, but its security is dubious. It is a fairly common practice to use IPsec instead.

In this case, part of your network may look like this:

```
West-----East == the rest of your network
workstation   untrusted wireless net
```

Of course, there would likely be several wireless workstations, each with its own IPsec tunnel to the East gateway.

The connection descriptions look much like Road Warrior descriptions:

- each workstation should have its own unique
  - ◆ identifier for IPsec
  - ◆ RSA key
  - ◆ connection description.
- on the gateway, use *left=%any*, or the workstation IP address
- on workstations, *left=%defaultroute*, or the workstation IP address
- *leftsubnet=* is not used.

The *rightsubnet=* parameter might be set in any of several ways:

```
rightsubnet=0.0.0.0/0
    allowing workstations to access the entire Internet (see above)
rightsubnet=a.b.c.0/24
    allowing access to your entire local network
rightsubnet=a.b.c.d/32
    restricting the workstation to connecting to a particular server
```

Of course you can mix and match these as required. For example, a university might allow faculty full Internet access while letting student laptops connect only to a group of lab machines.

## Choosing connection types

One choice you need to make before configuring additional connections is what type or types of connections you will use. There are several options, and you can use more than one concurrently.

## Manual vs. automatic keying

IPsec allows two types of connections, with manual or automatic keying. FreeS/WAN starts them with commands such as:

```
ipsec manual --up name
ipsec auto --up name
```

The difference is in how they are keyed.

### Manually keyed connections

use keys stored in ipsec.conf.

### Automatically keyed connections

use keys automatically generated by the Pluto key negotiation daemon. The key negotiation protocol, IKE, must authenticate the other system. (It is vulnerable to a man-in-the-middle attack if used without authentication.) We currently support two authentication methods:

- ◊ using shared secrets stored in ipsec.secrets.
- ◊ RSA public key authentication, with our machine's private key in ipsec.secrets. Public keys for other machines may either be placed in ipsec.conf or provided via DNS.

A third method, using RSA keys embedded in X.509 certificates, is provided by user patches.

Manually keyed connections provide weaker security than automatically keyed connections. An opponent who reads ipsec.secrets(5) gets your encryption key and can read all data encrypted by it. If he or she has an archive of old messages, all of them back to your last key change are also readable.

With automatically-(re)-keyed connections, an opponent who reads ipsec.secrets(5) gets the key used to authenticate your system in IKE — the shared secret or your private key, depending what authentication mechanism is in use. However, he or she does not automatically gain access to any encryption keys or any data.

An attacker who has your authentication key can mount a man-in-the-middle attack and, if that succeeds, he or she will get encryption keys and data. This is a serious danger, but it is better than having the attacker read everything as soon as he or she breaks into ipsec.secrets(5). Moreover, the keys change often so an opponent who gets one key does not get a large amount of data. To read all your data, he or she would have to do a man-in-the-middle attack at every key change.

We discuss using manual keying in production below, but this is ***not recommended*** except in special circumstances, such as needing to communicate with some implementation that offers no auto-keyed mode compatible with FreeS/WAN.

Manual keying may also be useful for testing. There is some discussion of this in our FAQ.

## Authentication methods for auto-keying

The IKE protocol which Pluto uses to negotiate connections between gateways must use some form of authentication of peers. A gateway must know who it is talking to before it can create a secure connection. We support two basic methods for this authentication:

- shared secrets, stored in ipsec.secrets(5)
- RSA authentication

There are, however, several variations on the RSA theme, using different methods of managing the RSA keys:

- our RSA private key in ipsec.secrets(5) with other gateways' public keys  
*either*  
stored in ipsec.conf(5)  
*or*  
looked up via DNS
- authentication with x.509 certificates.; See our links section for information on user-contributed patches for this.:

Public keys in ipsec.conf(5) give a reasonably straightforward method of specifying keys for explicitly configured connections.

Putting public keys in DNS allows us to support opportunistic encryption. Any two FreeS/WAN gateways can provide secure communication, without either of them having any preset information about the other.

X.509 certificates may be required to interface to various PKIs.

### Advantages of public key methods

Authentication with a public key method such as RSA has some important advantages over using shared secrets.

- no problem of secure transmission of secrets
  - ◆ A shared secret must be shared, so you have the problem of transmitting it securely to the other party. If you get this wrong, you have no security.
  - ◆ With a public key technique, you transmit only your public key. The system is designed to ensure that it does not matter if an enemy obtains public keys. The private key never leaves your machine.
- easier management
  - ◆ Suppose you have 20 branch offices all connecting to one gateway at head office, and all using shared secrets. Then the head office admin has 20 secrets to manage. Each of them must be kept secret not only from outsiders, but also from 19 of the branch office admins. The branch office admins have only one secret each to manage.

If the branch offices need to talk to each other, this becomes problematic. You need another  $20 \times 19/2 = 190$  secrets for branch-to-branch communication, each known to exactly two branches. Now all the branch admins have the headache of handling 20 keys, each shared with exactly one other branch or with head office.

For larger numbers of branches, the number of connections and secrets increases quadratically and managing them becomes a nightmare. A 1000-gateway fully connected network needs 499,500 secrets, each known to exactly two players. There are ways to reduce this problem, for example by introducing a central key server, but these involve additional communication overheads, more administrative work, and new threats that must be carefully guarded against.

- ◆ With public key techniques, the *only* thing you have to keep secret is your private key, and *you keep that secret from everyone*.

As network size increases, the number of public keys used increases linearly with the number

of nodes. This still requires careful administration in large applications, but is nothing like the disaster of a quadratic increase. On a 1000-gateway network, you have 1000 private keys, each of which must be kept secure on one machine, and 1000 public keys which must be distributed. This is not a trivial problem, but it is manageable.

- does not require fixed IP addresses
  - ◆ When shared secrets are used in IPsec, the responder must be able to tell which secret to use by looking at the IP address on the incoming packets. When the other parties do not have a fixed IP address to be identified by (for example, on nearly all dialup ISP connections and many cable or ADSL links), this does not work well — all must share the same secret!
  - ◆ When RSA authentication is in use, the initiator can identify itself by name before the key must be determined. The responder then checks that the message is signed with the public key corresponding to that name.

There is also a disadvantage:

- your private key is a single point of attack, extremely valuable to an enemy
  - ◆ with shared secrets, an attacker who steals your `ipsec.secrets` file can impersonate you or try man-in-the-middle attacks, but can only attack connections described in that file
  - ◆ an attacker who steals your private key gains the chance to attack not only existing connections *but also any future connections* created using that key

This is partly counterbalanced by the fact that the key is never transmitted and remains under your control at all times. It is likely necessary, however, to take account of this in setting security policy. For example, you should change gateway keys when an administrator leaves the company, and should change them periodically in any case.

Overall, public key methods are *more secure, more easily managed and more flexible*. We recommend that they be used for all connections, unless there is a compelling reason to do otherwise.

## Using shared secrets in production

Generally, public key methods are preferred for reasons given above, but shared secrets can be used with no loss of security, just more work and perhaps more need to take precautions.

What I call "shared secrets" are sometimes also called "pre-shared keys". They are used only for authentication, never for encryption. Calling them "pre-shared keys" has confused some users into thinking they were encryption keys, so I prefer to avoid the term..

If you are interoperating with another IPsec implementation, you may find its documentation calling them "passphrases".

### Putting secrets in `ipsec.secrets(5)`

If shared secrets are to be used to authenticate communication for the Diffie-Hellman key exchange in the IKE protocol, then those secrets must be stored in `/etc/ipsec.secrets`. For details, see the `ipsec.secrets(5)` man page.

A few considerations are vital:

## Introduction to FreeS/WAN

- make the secrets long and unguessable. Since they need not be remembered by humans, very long ugly strings may be used. We suggest using our ipsec\_ranbits(8) utility to generate long (128 bits or more) random strings.
- transmit secrets securely. You have to share them with other systems, but you lose if they are intercepted and used against you. Use PGP, SSH, hand delivery of a floppy disk which is then destroyed, or some other trustworthy method to deliver them.
- store secrets securely, in root-owned files with permissions `rw-----`.
- limit sharing of secrets. Alice, Bob, Carol and Dave may all talk to each other, but only Alice and Bob should know the secret for an Alice-Bob link.
- ***do not share private keys***. The private key for RSA authentication of your system is stored in ipsec.secrets(5), but it is a different class of secret from the pre-shared keys used for the "shared secret" authentication. No-one but you should have the RSA private key.

Each line has the IP addresses of the two gateways plus the secret. It should look something like this:

```
10.0.0.1 11.0.0.1 : PSK "jxTR11nmSjuj33n4W51uW3kTR551uUmSmnlRUuWnkjRj3UuTV4T3USSu23Uk55nWu
```

*PSK* indicates the use of a *pre-shared key*. The quotes and the whitespace shown are required.

You can use any character string as your secret. For security, it should be both long and extremely hard to guess. We provide a utility to generate such strings, ipsec\_ranbits(8).

You want the same secret on the two gateways used, so you create a line with that secret and the two gateway IP addresses. The installation process supplies an example secret, useful *only* for testing. You must change it for production use.

## File security

You must deliver this file, or the relevant part of it, to the other gateway machine by some *secure* means. *Don't just FTP or mail the file!* It is vital that the secrets in it remain secret. An attacker who knew those could easily have *all the data on your "secure" connection*.

This file must be owned by root and should have permissions `rw-----`.

## Shared secrets for road warriors

You can use a shared secret to support a single road warrior connecting to your gateway, and this is a reasonable thing to do in some circumstances. Public key methods have advantages, discussed above, but they are not critical in this case.

To do this, the line in ipsec.secrets(5) is something like:

```
10.0.0.1 0.0.0.0 : PSK "jxTR11nmSjuj33n4W51uW3kTR551uUmSmnlRUuWnkjRj3UuTV4T3USSu23Uk55nWu
```

where the *0.0.0.0* means that any IP address is acceptable.

***For more than one road warrior, shared secrets are not recommended.*** If shared secrets are used, then when the responder needs to look up the secret, all it knows about the sender is an IP address. This is fine if the sender is at a fixed IP address specified in the config file. It is also fine if only one road warrior uses the wildcard *0.0.0.0* address. However, if you have more than one road warrior using shared secret authentication, then they must all use that wildcard and therefore ***all road warriors using PSK authentication***

*must use the same secret.* Obviously, this is insecure.

*For multiple road warriors, use public key authentication.* Each roadwarrior can then have its own identity (our *leftid=* or *rightid=* parameters), its own public/private key pair, and its own secure connection.

## Using manual keying in production

Generally, automatic keying is preferred over manual keying for production use because it is both easier to manage and more secure. Automatic keying frees the admin from much of the burden of managing keys securely, and can provide perfect forward secrecy. This is discussed in more detail above.

However, it is possible to use manual keying in production if that is what you want to do. This might be necessary, for example, in order to interoperate with some device that either does not provide automatic keying or provides it in some version we cannot talk to.

Note that with manual keying *all security rests with the keys*. If an adversary acquires your keys, you've had it. He or she can read everything ever sent with those keys, including old messages he or she may have archived.

You need to *be really paranoid about keys* if you're going to rely on manual keying for anything important.

- keep keys in files with 600 permissions, owned by root
- be extremely careful about security of your gateway systems. Anyone who breaks into a gateway and gains root privileges can get all your keys and read everything ever encrypted with those keys, both old messages he has archived and any new ones you may send.
- change keys regularly. This can be a considerable bother, (and provides an excellent reason to consider automatic keying instead), but it is *absolutely essential* for security. Consider a manually keyed system in which you leave the same key in place for months:
  - ◆ an attacker can have a very large sample of text sent with that key to work with. This makes various cryptographic attacks much more likely to succeed.
  - ◆ The chances of the key being compromised in some non-cryptographic manner — a spy finds it on a discarded notepad, someone breaks into your server or your building and steals it, a staff member is bribed, tricked, seduced or coerced into revealing it, etc. — also increase over time.
  - ◆ a successful attacker can read everything ever sent with that key. This makes any successful attack extremely damaging.

It is clear that you must change keys often to have any useful security. The only question is how often.

- use PGP or SSH for all key transfers
- don't edit files with keys in them when someone can look over your shoulder
- worry about network security; could someone get keys by snooping packets on the LAN between your X desktop and the gateway?
- lock up your backup tapes for the gateway system
- ... and so on

Linux FreeS/WAN provides some facilities to help with this. In particular, it is good policy to *keep keys in separate files* so you can edit configuration information in `/etc/ipsec.conf` without exposing keys to "shoulder surfers" or network snoops. We support this with the *also=* and *include* syntax in `ipsec.conf(5)`.

See the last example in our `examples` file. In the `/etc/ipsec.conf` *conn samplesep* section, it has the line:

## Introduction to FreeS/WAN

```
also=samplesep-keys
```

which tells the "ipsec manual" script to insert the configuration description labelled "samplesep-keys" if it can find it. The /etc/ipsec.conf file must also have a line such as:

```
include ipsec.*.conf
```

which tells it to read other files. One of those other files then might contain the additional data:

```
conn samplesep-keys
    spi=0x200
    esp=3des-md5-96
    espenckey=0x01234567_89abcdef_02468ace_13579bdf_12345678_9abcdef0
    espauthkey=0x12345678_9abcdef0_2468ace0_13579bdf
```

The first line matches the label in the "also=" line, so the indented lines are inserted. The net effect is exactly as if the inserted lines had occurred in the original file in place of the "also=" line.

Variables set here are:

*spi*

A number needed by the manual keying code. Any 3-digit hex number will do, but if you have more than one manual connection then *spi must be different* for each connection.

*esp*

Options for ESP (Encapsulated Security Payload), the usual IPsec encryption mode. Settings here are for encryption using triple DES and authentication using MD5. Note that encryption without authentication should not be used; it is insecure.

*espenkey*

Key for ESP encryption. Here, a 192-bit hex number for triple DES.

*espauthkey*

Key for ESP authentication. Here, a 128-bit hex number for MD5.

**Note** that the *example keys we supply* are intended *only for testing*. For real use, you should go to automatic keying. If that is not possible, create your own keys for manual mode and keep them secret

Of course, any files containing keys *must* have 600 permissions and be owned by root.

If you connect in this way to multiple sites, we recommend that you keep keys for each site in a separate file and adopt some naming convention that lets you pick them all up with a single "include" line. This minimizes the risk of losing several keys to one error or attack and of accidentally giving another site admin keys which he or she has no business knowing.

Also note that if you have multiple manually keyed connections on a single machine, then the *spi* parameter must be different for each one. Any 3-digit hex number is OK, provided they are different for each connection. We reserve the range 0x100 to 0xffff for manual connections. Pluto assigns SPIs from 0x1000 up for automatically keyed connections.

If ipsec.conf(5) contains keys for manual mode connections, then it too must have permissions *rw-----*. We recommend instead that, if you must manual keying in production, you keep the keys in separate files.

Note also that ipsec.conf is installed with permissions *rw-r--r--*. If you plan to use manually keyed connections for anything more than initial testing, you **must**:

- either change permissions to *rw-----*
- or store keys separately in secure files and access them via include statements in ipsec.conf.

We recommend the latter method for all but the simplest configurations.

## Creating keys with ranbits

You can create new random keys with the ranbits(8) utility. For example, the commands:

```
umask 177
ipsec ranbits 192 > temp
ipsec ranbits 128 >> temp
```

create keys in the sizes needed for our default algorithms:

- 192-bit key for 3DES encryption  
(only 168 bits are used; parity bits are ignored)
- 128-bit key for keyed MD5 authentication

If you want to use SHA instead of MD5, that requires a 160-bit key

Note that any *temporary files* used must be kept *secure* since they contain keys. That is the reason for the umask command above. The temporary file should be deleted as soon as you are done with it. You may also want to change the umask back to its default value after you are finished working on keys.

The ranbits utility may pause for a few seconds if not enough entropy is available immediately. See ipsec\_ranbits(8) and random(4) for details. You may wish to provide some activity to feed entropy into the system. For example, you might move the mouse around, type random characters, or do *du /usr > /dev/null* in the background.

## Setting up connections at boot time

You can tell the system to set up connections automatically at boot time by putting suitable stuff in */etc/ipsec.conf* on both systems. The relevant section of the file is labelled by a line reading *config setup*.

Details can be found in the ipsec.conf(5) man page. We also provide a file of example configurations.

The most likely options are something like:

```
interfaces="ipsec0=eth0 ipsec1=ppp0"
```

Tells KLIPS which interfaces to use. Up to four interfaces numbered ipsec[0–3] are supported. Each interface can support an arbitrary number of tunnels.

Note that for PPP, you give the ppp[0–9] device name here, not the underlying device such as modem (or eth1 if you are using PPPoE).

```
interfaces=%defaultroute
```

Alternative setting, useful in simple cases. KLIPS will pick up both its interface and the next hop information from the settings of the Linux default route.

```
forwardcontrol=no
```

Normally "no". Set to "yes" if the IP forwarding option is disabled in your network configuration.

## Introduction to FreeS/WAN

(This can be set as a kernel configuration option or later. e.g. on Redhat, it's in /etc/sysconfig/network and on SuSE you can adjust it with Yast.) Linux FreeS/WAN will then enable forwarding when starting up and turn it off when going down. This is used to ensure that no packets will be forwarded before IPsec comes up and takes control.

*syslog=daemon.error*

Used in messages to the system logging daemon (syslogd) to specify what type of software is sending the messages. If the settings are "daemon.error" as in our example, then syslogd treats the messages as error messages from a daemon.

Note that Pluto does not currently pay attention to this variable. The variable controls setup messages only.

*klipsdebug=*

Debug settings for KLIPS.

*plutodebug=*

Debug settings for Pluto.

*... for both the above DEBUG settings*

Normally, leave empty as shown above for no debugging output.

Use "all" for maximum information.

See ipsec\_klipsdebug(8) and ipsec\_pluto(8) man page for other options. Beware that if you set /etc/ipsec.conf to enable debug output, your system's log files may get large quickly.

*dumpdir=/safe/directory*

Normally, programs started by ipsec setup don't crash. If they do, by default, no core dump will be produced because such dumps would contain secrets. If you find you need to debug such crashes, you can set dumpdir to the name of a directory in which to collect the core file.

*manualstart=*

List of manually keyed connections to be automatically started at boot time. Useful for testing, but not for long term use. Connections which are automatically started should also be automatically re-keyed.

*pluto=yes*

Whether to start Pluto when ipsec startup is done.

This parameter is optional and defaults to "yes" if not present.

"yes" is strongly recommended for production use so that the keying daemon (Pluto) will automatically re-key the connections regularly. The ipsec-auto parameters ikelifetime, ipseclifetime and rekeywindow give you control over frequency of rekeying.

*plutoload="reno-van reno-adam reno-nyc"*

List of tunnels (by name, e.g. fred-susan or reno-van in our examples) to be loaded into Pluto's internal database at startup. In this example, Pluto loads three tunnels into its database when it is started.

If plutoload is "%search", Pluto will load any connections whose description includes "auto=add" or "auto=start".

*plutostart="reno-van reno-adam reno-nyc"*

List of tunnels to attempt to negotiate when Pluto is started.

If plutostart is "%search", Pluto will start any connections whose description includes "auto=start".

Note that, for a connection intended to be permanent, **both gateways should be set try to start** the tunnel. This allows quick recovery if either gateway is rebooted or has its IPsec restarted. If only one gateway is set to start the tunnel and the other gateway restarts, the tunnel may not be rebuilt.

*plutowait=no*

Controls whether Pluto waits for one tunnel to be established before starting to negotiate the next. You might set this to "yes"

◊ if your gateway is a very limited machine and you need to conserve resources.

◊ for debugging; the logs are clearer if only one connection is brought up at a time

For a busy and resource-laden production gateway, you likely want "no" so that connections are brought up in parallel and the whole process takes less time.

The example assumes you are at the Reno office and will use IPsec to Vancouver, New York City and Amsterdam.

## Multiple tunnels between the same two gateways

Consider a pair of subnets, each with a security gateway, connected via the Internet:

```
192.168.100.0/24      left subnet
|
192.168.100.1
North Gateway
101.101.101.101      left
|
101.101.101.1        left next hop
[Internet]
202.202.202.1        right next hop
|
202.202.202.202      right
South gateway
192.168.200.1
|
192.168.200.0/24     right subnet
```

A tunnel specification such as:

```
conn northnet-southnet
    left=101.101.101.101
    leftnexthop=101.101.101.1
    leftsubnet=192.168.100.0/24
    leftfirewall=yes
    right=202.202.202.202
    rightnexthop=202.202.202.1
    rightsubnet=192.168.200.0/24
    rightfirewall=yes
```

will allow machines on the two subnets to talk to each other. You might test this by pinging from polarbear (192.168.100.7) to penguin (192.168.200.5).

However, *this does not cover other traffic you might want to secure*. To handle all the possibilities, you might also want these connection descriptions:

```
conn northgate-southnet
    left=101.101.101.101
    leftnexthop=101.101.101.1
    right=202.202.202.202
    rightnexthop=202.202.202.1
    rightsubnet=192.168.200.0/24
    rightfirewall=yes
```

## Introduction to FreeS/WAN

```
conn northnet-southgate
    left=101.101.101.101
    leftnexthop=101.101.101.1
    leftsubnet=192.168.100.0/24
    leftfirewall=yes
    right=202.202.202.202
    rightnexthop=202.202.202.1
```

Without these, neither gateway can do IPsec to the remote subnet. There is no IPsec tunnel or eroute set up for the traffic.

In our example, with the non-routable 192.168.\* addresses used, packets would simply be discarded. In a different configuration, with routable addresses for the remote subnet, *they would be sent unencrypted* since there would be no IPsec eroute and there would be a normal IP route.

You might also want:

```
conn northgate-southgate
    left=101.101.101.101
    leftnexthop=101.101.101.1
    right=202.202.202.202
    rightnexthop=202.202.202.1
```

This is required if you want the two gateways to speak IPsec to each other.

This requires a lot of duplication of details. Judicious use of *also=* and *include* can reduce this problem.

Note that, while FreeS/WAN supports all four tunnel types, not all implementations do. In particular, some versions of Windows 2000 and the freely downloadable version of PGP provide only "client" functionality. You cannot use them as gateways with a subnet behind them. To get that functionality, you must upgrade to Windows 2000 server or the commercially available PGP products.

## One tunnel plus advanced routing

It is also possible to use the new routing features in 2.2 and later kernels to avoid most needs for multiple tunnels. Here is one mailing list message on the topic:

```
Subject: Re: linux-ipsec: IPSec packets not entering tunnel?
Date: Mon, 20 Nov 2000
From: Justin Guyett <jfg@sonicity.com>
```

On Mon, 20 Nov 2000, Claudia Schmeing wrote:

```
> Right                                                    Left
>
>                "home"                "office"
> 10.92.10.0/24 ---- 24.93.85.110 ===== 216.175.164.91 ---- 10.91.10.24/24
>
> I've created all four tunnels, and can ping to test each of them,
> *except* homegate-officenet.
```

I keep wondering why people create all four tunnels. Why not route traffic generated from home to 10.91.10.24/24 out ipsec0 with iproute2? And 99% of the time you don't need to access "office" directly, which means you can eliminate all but the subnet<->subnet connection.

and FreeS/WAN technical lead Henry Spencer's comment:

## Introduction to FreeS/WAN

```
> I keep wondering why people create all four tunnels. Why not route
> traffic generated from home to 10.91.10.24/24 out ipsec0 with iproute2?
```

This is feasible, given some iproute2 attention to source addresses, but it isn't something we've documented yet... (partly because we're still making some attempt to support 2.0.xx kernels, which can't do this, but mostly because we haven't caught up with it yet).

```
> And 99% of the time you don't need to access "office" directly, which
> means you can eliminate all but the subnet<->subnet connection.
```

Correct in principle, but people will keep trying to ping to or from the gateways during testing, and sometimes they want to run services on the gateway machines too.

## An Opportunistic Gateway

### Start from full opportunism

Full opportunism allows you to initiate and receive opportunistic connections on your machine. The remaining instructions in this section assume you have first set up full opportunism on your gateway using [these instructions](#). Both sets of instructions require mailing DNS records to your ISP. Collect DNS records for both the gateway (above) and the subnet nodes (below) before contacting your ISP.

### Reverse DNS TXT records for each protected machine

You need these so that your Opportunistic peers can:

- discover the gateway's address, knowing only the IP address that packets are bound for
- verify that the gateway is authorised to encrypt for that endpoint

On the gateway, generate a TXT record with:

```
ipsec showhostkey --txt 192.0.2.11
```

Use your gateway address in place of 192.0.2.11.

You should see (keys are trimmed for clarity throughout our example):

```
; RSA 2048 bits gateway.example.com Sat Apr 15 13:53:22 2000
IN TXT "X-IPsec-Server(10)=192.0.2.11" " AQOF8tZ2...+buFuFn/"
```

**This MUST BE the same key as in your gateway's TXT record, or nothing will work.**

In a text file, make one copy of this TXT record for each subnet node:

```
; RSA 2048 bits gateway.example.com Sat Apr 15 13:53:22 2000
IN TXT "X-IPsec-Server(10)=192.0.2.11" " AQOF8tZ2...+buFuFn/"
```

```
; RSA 2048 bits gateway.example.com Sat Apr 15 13:53:22 2000
IN TXT "X-IPsec-Server(10)=192.0.2.11" " AQOF8tZ2...+buFuFn/"
```

```
; RSA 2048 bits gateway.example.com Sat Apr 15 13:53:22 2000
IN TXT "X-IPsec-Server(10)=192.0.2.11" " AQOF8tZ2...+buFuFn/"
```

## Introduction to FreeS/WAN

Above each entry, insert a line like this:

```
98.2.0.192.in-addr.arpa. IN PTR arthur.example.com.
```

It must include:

- The subnet node's address in reverse map format. For example, 192.0.2.120 becomes *120.2.0.192.in-addr.arpa.* Note the final period.
- *IN PTR*
- The node's name, ie. *arthur.example.com.* Note the final period.

The result will be a file of TXT records, like this:

```
98.2.0.192.in-addr.arpa. IN PTR arthur.example.com.
; RSA 2048 bits gateway.example.com Sat Apr 15 13:53:22 2000
IN TXT "X-IPsec-Server(10)=192.0.2.11" " AQOF8tZ2...+buFuFn/"

99.2.0.192.in-addr.arpa. IN PTR ford.example.com.
; RSA 2048 bits gateway.example.com Sat Apr 15 13:53:22 2000
IN TXT "X-IPsec-Server(10)=192.0.2.11" " AQOF8tZ2...+buFuFn/"

100.2.0.192.in-addr.arpa. IN PTR trillian.example.com.
; RSA 2048 bits gateway.example.com Sat Apr 15 13:53:22 2000
IN TXT "X-IPsec-Server(10)=192.0.2.11" " AQOF8tZ2...+buFuFn/"
```

## Publish your records

Ask your ISP to publish all the reverse DNS records you have collected. There may be a delay of up to 48 hours as the records propagate.

## ...and test them

Check a couple of records with commands like this one:

```
ipsec verify --host ford.example.com
ipsec verify --host trillian.example.com
```

The *verify* command checks for TXT records for both the subnet host and its gateway. You should see output like:

```
...
Looking for TXT in reverse map: 99.2.0.192.in-addr.arpa [OK]
...
Looking for TXT in reverse map: 11.2.0.192.in-addr.arpa [OK]
...
Looking for TXT in reverse map: 100.2.0.192.in-addr.arpa [OK]
...
Looking for TXT in reverse map: 11.2.0.192.in-addr.arpa [OK]
...
```

## No Configuration Needed

FreeS/WAN 2.x ships with a built-in, automatically enabled OE connection *conn packetdefault* which applies OE, if possible, to all outbound traffic routed through the FreeS/WAN box. The [ipsec.conf\(5\) manual](#)

describes this connection in detail. While the effect is much the same as *private-or-clear*, the implementation is different: notably, it does not use policy groups.

You can create more complex OE configurations for traffic forwarded through a FreeS/WAN box, as explained in our [policy groups document](#), or disable OE using [these instructions](#).

## Extruded Subnets

What we call [extruded subnets](#) are a special case of [VPNs](#).

If your buddy has some unused IP addresses, in his subnet far off at the other side of the Internet, he can loan them to you... provided that the connection between you and him is fast enough to carry all the traffic between your machines and the rest of the Internet. In effect, he "extrudes" a part of his address space over the network to you, with your Internet traffic appearing to originate from behind his Internet gateway.

As far as the Internet is concerned, your new extruded net is behind your buddy's gateway. You route all your packets for the Internet at large out his gateway, and receive return packets the same way. You route your local packets locally.

Suppose your friend has a.b.c.0/24 and wants to give you a.b.c.240/28. The initial situation is:

subnet	gateway	Internet
a.b.c.0/24	a.b.c.1	p.q.r.s

where anything from the Internet destined for any machine in a.b.c.0/24 is routed via p.q.r.s and that gateway knows what to do from there.

Of course it is quite normal for various smaller subnets to exist behind your friend's gateway. For example, your friend's company might have a.b.c.16/28=development, a.b.c.32/28=marketing and so on. The Internet neither knows nor cares about this; it just delivers packets to the p.q.r.s and lets the gateway do whatever needs to be done from there.

What we want to do is take a subnet, perhaps a.b.c.240/28, out of your friend's physical location *while still having your friend's gateway route to it*. As far as the Internet is concerned, you remain behind that gateway.

subnet	gateway	Internet	your gate	extruded
a.b.c.0/24	a.b.c.1	p.q.r.s	d.e.f.g	a.b.c.240/28
===== tunnel =====				

The extruded addresses have to be a complete subnet.

In our example, the friend's security gateway is also his Internet gateway, but this is not necessary. As long as all traffic from the Internet to his addresses passes through the Internet gate, the security gate could be a machine behind that. The IG would need to route all traffic for the extruded subnet to the SG, and the SG could handle the rest.

First, configure your subnet using the extruded addresses. Your security gateway's interface to your subnet needs to have an extruded address (possibly using a Linux [virtual interface](#), if it also has to have a different address). Your gateway needs to have a route to the extruded subnet, pointing to that interface. The other

## Introduction to FreeS/WAN

machines at your site need to have addresses in that subnet, and default routes pointing to your gateway.

If any of your friend's machines need to talk to the extruded subnet, *they* need to have a route for the extruded subnet, pointing at his gateway.

Then set up an IPsec subnet-to-subnet tunnel between your gateway and his, with your subnet specified as the extruded subnet, and his subnet specified as "0.0.0.0/0".

The tunnel description should be:

```
conn extruded
    left=p.q.r.s
    leftsubnet=0.0.0.0/0
    right=d.e.f.g
    rightsubnet=a.b.c.0/28
```

If either side was doing firewalling for the extruded subnet before the IPsec connection is set up, you'll need to poke holes in your firewall to allow packets through.

And it all just works. Your SG routes traffic for 0.0.0.0/0 — that is, the whole Internet — through the tunnel to his SG, which then sends it onward as if it came from his subnet. When traffic for the extruded subnet arrives at his SG, it gets sent through the tunnel to your SG, which passes it to the right machine.

Remember that when `ipsec_manual` or `ipsec_auto` takes a connection down, it *does not undo the route* it made for that connection. This lets you take a connection down and bring up a new one, or a modified version of the old one, without having to rebuild the route it uses and without any risk of packets which should use IPsec accidentally going out in the clear. Because the route always points into KLIPS, the packets will always go there. Because KLIPS temporarily has no idea what to do with them (no eroute for them), they will be discarded.

If you *do* want to take the route down, this is what the "unroute" operation in manual and auto is for. Just do an unroute after doing the down.

Note that the route for a connection may have replaced an existing non-IPsec route. Nothing in Linux FreeS/WAN will put that pre-IPsec route back. If you need it back, you have to create it with the route command.

## Road Warrior with virtual IP address

Please note that Super FreeS/WAN now features DHCP-over-IPsec, which is an alternate procedure for Virtual IP address assignment.

Here is a mailing list message about another way to configure for road warrior support:

```
Subject: Re: linux-ipsec: understanding the vpn
Date: Thu, 28 Oct 1999 10:43:22 -0400
From: Irving Reid <irving@nevex.com>

> local-----linux-----internet-----mobile
> LAN          box                      user
> ...

> now when the mobile user connects to the linux box
```

## Introduction to FreeS/WAN

```
> it is given a virtual IP address, i have configured it to
> be in the 10.x.x.x range. mobile user and linux box
> have a tunnel between them with these IP addresses.

> Uptil this all is fine.
```

If it is possible to configure your mobile client software *\*not\** to use a virtual IP address, that will make your life easier. It is easier to configure FreeS/WAN to use the actual address the mobile user gets from its ISP.

Unfortunately, some Windows clients don't let you choose.

```
> what i would like to know is that how does the mobile
> user communicate with other computers on the local
> LAN , of course with the vpn ?

> what IP address should the local LAN
> computers have ? I guess their default gateway
> should be the linux box ? and does the linux box need
> to be a 2 NIC card box or one is fine.
```

As someone else stated, yes, the Linux box would usually be the default IP gateway for the local lan.

However...

If you mobile user has software that *\*must\** use a virtual IP address, the whole picture changes. Nobody has put much effort into getting FreeS/WAN to play well in this environment, but here's a sketch of one approach:

```
Local Lan 1.0.0.0/24
|
+- Linux FreeS/WAN 1.0.0.2
|
| 1.0.0.1
Router
| 2.0.0.1
|
Internet
|
| 3.0.0.1
Mobile User
    Virtual Address: 1.0.0.3
```

Note that the Local Lan network (1.0.0.x) can be registered, routable addresses.

Now, the Mobile User sets up an IPSec security association with the Linux box (1.0.0.2); it should ESP encapsulate all traffic to the network 1.0.0.x **\*\*EXCEPT\*\*** UDP port 500. 500/udp is required for the key negotiation, which needs to work outside of the IPSec tunnel.

On the Linux side, there's a bunch of stuff you need to do by hand (for now). FreeS/WAN should correctly handle setting up the IPSec SA and routes, but I haven't tested it so this may not work...

The FreeS/WAN conn should look like:

```
conn mobile
    right=1.0.0.2
```

## Introduction to FreeS/WAN

```
rightsubnet=1.0.0.0/24
rightnexthop=1.0.0.1
left=0.0.0.0 # The infamous "road warrior"
leftsubnet=1.0.0.3/32
```

Note that the left subnet contains *only* the remote host's virtual address.

Hopefully the routing table on the FreeS/WAN box ends up looking like this:

```
% netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask          Flags      MSS Window  irtt Iface
1.0.0.0          0.0.0.0          255.255.255.0    U          1500 0         0 eth0
127.0.0.0        0.0.0.0          255.0.0.0        U          3584 0         0 lo
0.0.0.0          1.0.0.1          0.0.0.0          UG         1500 0         0 eth0
1.0.0.3          1.0.0.1          255.255.255.255 UG         1433 0         0 ipsec0
```

So, if anybody sends a packet for 1.0.0.3 to the Linux box, it should get bundled up and sent through the tunnel. To get the packets for 1.0.0.3 to the Linux box in the first place, you need to use "proxy ARP".

How this works is: when a host or router on the local Ethernet segment wants to send a packet to 1.0.0.3, it sends out an Ethernet level broadcast "ARP request". If 1.0.0.3 was on the local LAN, it would reply, saying "send IP packets for 1.0.0.3 to my Ethernet address".

Instead, you need to set up the Linux box so that *it* answers ARP requests for 1.0.0.3, even though that isn't its IP address. That convinces everyone else on the lan to send 1.0.0.3 packets to the Linux box, where the usual FreeS/WAN processing and routing take over.

```
% arp -i eth0 -s 1.0.0.3 -D eth0 pub
```

This says, if you see an ARP request on interface eth0 asking for 1.0.0.3, respond with the Ethernet address of interface eth0.

Now, as I said at the very beginning, if it is *at all* possible to configure your client *not* to use the virtual IP address, you can avoid this whole mess.

## Dynamic Network Interfaces

Sometimes you have to cope with a situation where the network interface(s) aren't all there at boot. The common example is notebooks with PCMCIA.

### Basics

The key issue here is that the *config setup* section of the */etc/ipsec.conf* configuration file lists the connection between ipsecN and hardware interfaces, in the *interfaces=* variable. At any time when *ipsec setup start* or *ipsec setup restart* is run this variable *must* correspond to the current real situation. More precisely, it *must not* mention any hardware interfaces which don't currently exist. The difficulty is that an *ipsec setup start* command is normally run at boot time so interfaces that are not up then are mis-handled.

## Boot Time

Normally, an *ipsec setup start* is run at boot time. However, if the hardware situation at boot time is uncertain, one of two things must be done.

- One possibility is simply not to have IPsec brought up at boot time. To do this:

```
chkconfig --level 2345 ipsec off
```

That's for modern Red Hats or other Linuxes with chkconfig. Systems which lack this will require fiddling with symlinks in `/etc/rc.d/rc?.d` or the equivalent.

- Another possibility is to bring IPsec up with no interfaces, which is less aesthetically satisfying but simpler. Just put

```
interfaces=
```

in the configuration file. KLIPS and Pluto will be started, but won't do anything.

## Change Time

When the hardware *\*is\** in place, IPsec has to be made aware of it. Someday there may be a nice way to do this.

Right now, the way to do it is to fix the `/etc/ipsec.conf` file appropriately, so *interfaces* reflects the new situation, and then restart the IPsec subsystem. This does break any existing IPsec connections.

If IPsec wasn't brought up at boot time, do

```
ipsec setup start
```

while if it was, do

```
ipsec setup restart
```

which won't be as quick.

If some of the hardware is to be taken out, before doing that, amend the configuration file so *interfaces* no longer includes it, and do

```
ipsec setup restart
```

Again, this breaks any existing connections.

## Unencrypted tunnels

Sometimes you might want to create a tunnel without encryption. Often this is a bad idea, even if you have some data which need not be private. See this [discussion](#).

The IPsec protocols provide two ways to do build such tunnels:

*using ESP with null encryption*

not supported by FreeS/WAN  
using AH without ESP  
supported for manually keyed connections  
possible with explicit commands via ipsec whack(8) (see this list message)  
not supported in the ipsec auto(8) scripts.

One situation in which this comes up is when otherwise some data would be encrypted twice. Alice wants a secure tunnel from her machine to Bob's. Since she's behind one security gateway and he's behind another, part of the tunnel that they build passes through the tunnel that their site admins have built between the gateways. All of Alice and Bob's messages are encrypted twice.

There are several ways to handle this.

- Just accept the overhead of double encryption. The site admins might choose this if any of the following apply:
  - ◆ policy says encrypt everything (usually, it should)
  - ◆ they don't entirely trust Alice and Bob (usually, if they don't have to, they shouldn't)
  - ◆ if they don't feel the saved cycles are worth the time they'd need to build a non-encrypted tunnel for Alice and Bob's packets (often, they aren't)
- Use a plain IP-in-IP tunnel. These are not well documented. A good starting point is in the Linux kernel source tree, in /usr/src/linux/drivers/net/README.tunnel.
- Use a manually-keyed AH-only tunnel.

Note that if Alice and Bob want end-to-end security, they must build a tunnel end-to-end between their machines or use some other end-to-end tool such as PGP or SSL that suits their data. The only question is whether the admins build some special unencrypted tunnel for those already-encrypted packets.

# Installing FreeS/WAN

This document will teach you how to install Linux FreeS/WAN. If your distribution comes with Linux FreeS/WAN, we offer tips to get you started.

## Requirements

To install FreeS/WAN you must:

- be running Linux with the 2.4 or 2.2 kernel series. See this [kernel compatibility table](#). We also have experimental support for 2.6 kernels. Here are two basic approaches:
  - ♦ install FreeS/WAN, including its [KLIPS](#) kernel code. This will remove the native IPsec stack and replace it with KLIPS.
  - ♦ install the FreeS/WAN [userland tools](#) (keying daemon and supporting scripts) for use with [2.6 kernel native IPsec](#).
- See also these [known issues with 2.6](#).
- have root access to your Linux box
- choose the version of FreeS/WAN you wish to install based on [mailing list reports](#)

## Choose your install method

There are three basic ways to get FreeS/WAN onto your system:

- activating and testing a FreeS/WAN that [shipped with your Linux distribution](#)
- [RPM install](#)
- [Install from source](#)

## FreeS/WAN ships with some Linuxes

FreeS/WAN comes with [these distributions](#).

If you're running one of these, include FreeS/WAN in the choices you make during installation, or add it later using the distribution's tools.

## FreeS/WAN may be altered...

Your distribution may have integrated extra features, such as Andreas Steffen's X.509 patch, into FreeS/WAN. It may also use custom startup script locations or directory names.

## You might need to create an authentication keypair

If your FreeS/WAN came with your distribution, you may wish to generate a fresh RSA key pair. FreeS/WAN will use these keys for authentication.

To do this, become root, and type:

```
ipsec newhostkey --output /etc/ipsec.secrets --hostname xy.example.com
```

## Introduction to FreeS/WAN

```
chmod 600 /etc/ipsec.secrets
```

where you replace xy.example.com with your machine's fully-qualified domain name. Generate some randomness, for example by wiggling your mouse, to speed the process.

The resulting ipsec.secrets looks like:

```
: RSA {
# RSA 2192 bits   xy.example.com   Sun Jun 8 13:42:19 2003
# for signatures only, UNSAFE FOR ENCRYPTION
#pubkey=0sAQOFppfeE3cC7wqJi...
Modulus: 0x85a697de137702ef0...
# everything after this point is secret
PrivateExponent: 0x16466ea5033e807...
Prime1: 0xdfb5003c8947b7cc88759065...
Prime2: 0x98f199b9149fde11ec956c814...
Exponent1: 0x9523557db0da7a885af90aee...
Exponent2: 0x65f6667b63153eb69db8f300dbb...
Coefficient: 0x90ad00415d3ca17bebff123413fc518...
}
# do not change the indenting of that "}"
```

In the actual file, the strings are much longer.

## Start and test FreeS/WAN

You can now start FreeS/WAN and test whether it's been successfully installed.

## RPM install

These instructions are for a recent Red Hat with a stock Red Hat kernel. We know that Mandrake and SUSE also produce FreeS/WAN RPMs. If you're running either, install using your distribution's tools.

## Download RPMs

Decide which functionality you need:

- standard FreeS/WAN RPMs. Use these shortcuts:
  - ◆ (for 2.6 kernels: userland only)  
ncftpget ftp://ftp.xs4all.nl/pub/crypto/freeswan/binaries/RedHat-RPMs/\*userland\*
  - ◆ (for 2.4 kernels)  
ncftpget ftp://ftp.xs4all.nl/pub/crypto/freeswan/binaries/RedHat-RPMs/^uname -r | tr -d 'a-wy-z'^/\*
  - ◆ or view all the offerings at our [FTP site](#).
- unofficial Super FreeS/WAN RPMs, which include Andreas Steffen's X.509 patch and more. Super FreeS/WAN RPMs do not currently include Network Address Translation (NAT) traversal, but Super FreeS/WAN source does.

For 2.6 kernels, get the latest FreeS/WAN userland RPM, for example:

```
freeswan-userland-2.03.9-0.i386.rpm
```

## Start and test FreeS/WAN

## Introduction to FreeS/WAN

Note: FreeS/WAN's support for 2.6 kernel IPsec is preliminary. Please see [2.6.known-issues](#), and the latest [mailing list reports](#).

Change to your new FreeS/WAN directory, and make and install the

For 2.4 kernels, get both kernel and userland RPMs. Check your kernel version with

```
uname -r
```

Get a kernel module which matches that version. For example:

```
freeswan-module-2.03_2.4.20_20.9-0.i386.rpm
```

Note: These modules **will only work on the Red Hat kernel they were built for**, since they are very sensitive to small changes in the kernel.

Get FreeS/WAN utilities to match. For example:

```
freeswan-userland-2.03_2.4.20_20.9-0.i386.rpm
```

## For freeswan.org RPMs: check signatures

While you're at our ftp site, grab the RPM signing key

```
freeswan-rpmsign.asc
```

If you're running RedHat 8.x or later, import this key into the RPM database:

```
rpm --import freeswan-rpmsign.asc
```

For RedHat 7.x systems, you'll need to add it to your PGP keyring:

```
pgp -ka freeswan-rpmsign.asc
```

Check the digital signatures on both RPMs using:

```
rpm --checksig freeswan*.rpm
```

You should see that these signatures are good:

```
freeswan-module-2.03_2.4.20_20.9-0.i386.rpm: pgp md5 OK  
freeswan-userland-2.03_2.4.20_20.9-0.i386.rpm: pgp md5 OK
```

## Install the RPMs

Become root:

```
su
```

For a first time install, use:

```
rpm -ivh freeswan*.rpm
```

For freeswan.org RPMs: check signatures

## Introduction to FreeS/WAN

To upgrade existing RPMs (and keep all .conf files in place), use:

```
rpm -Uvh freeswan*.rpm
```

If you're upgrading from FreeS/WAN 1.x to 2.x RPMs, and encounter problems, see [this note](#).

## Start and Test FreeS/WAN

Now, [start FreeS/WAN and test your install](#).

## Install from Source

### Decide what functionality you need

Your choices are:

- [standard FreeS/WAN](#),
- standard FreeS/WAN plus any of these [user-supported patches](#), or
- [Super FreeS/WAN](#), an unofficial FreeS/WAN pre-patched with many of the above. Provides additional algorithms, X.509, SA deletion, dead peer detection, and [Network Address Translation](#) (NAT) traversal.

### Download FreeS/WAN

Download the source tarball you've chosen, along with any patches.

### For freeswan.org source: check its signature

While you're at our ftp site, get our source signing key

```
freeswan-sigkey.asc
```

Add it to your PGP keyring:

```
pgp -ka freeswan-sigkey.asc
```

Check the signature using:

```
pgp freeswan-2.03.tar.gz.sig freeswan-2.03.tar.gz
```

You should see something like:

```
Good signature from user "Linux FreeS/WAN Software Team (build@freeswan.org)".  
Signature made 2002/06/26 21:04 GMT using 2047-bit key, key ID 46EAFCE1
```

### Untar, unzip

As root, unpack your FreeS/WAN source into */usr/src*.

```
su
mv freeswan-2.03.tar.gz /usr/src
cd /usr/src
tar -xzf freeswan-2.03.tar.gz
```

### Patch if desired

Now's the time to add any patches. The contributor may have special instructions, or you may simply use the patch command.

### ... and Make

Choose one of the methods below.

#### Userland-only Install for 2.6 kernels

Note: FreeS/WAN's support for 2.6 kernel IPsec is preliminary. Please see [2.6.known-issues](#), and the latest [mailing list reports](#).

Change to your new FreeS/WAN directory, and make and install the FreeS/WAN userland tools.

```
cd /usr/src/freeswan-2.03
make programs
make install
```

Now, [start FreeS/WAN and test your install](#).

#### KLIPS install for 2.2, 2.4, or 2.6 kernels

To make a modular version of KLIPS, along with other FreeS/WAN programs you'll need, use the command sequence below. This will change to your new FreeS/WAN directory, make the FreeS/WAN module (and other stuff), and install it all.

```
cd /usr/src/freeswan-2.03
make oldmod
make minstall
```

[Start FreeS/WAN and test your install](#).

To link KLIPS statically into your kernel (using your old kernel settings), and install other FreeS/WAN components, do:

```
cd /usr/src/freeswan-2.03
make oldmod
make minstall
```

Reboot your system and [test your install](#).

For other ways to compile KLIPS, see our Makefile.

## Start FreeS/WAN and test your install

Bring FreeS/WAN up with:

```
service ipsec start
```

This is not necessary if you've rebooted.

## Test your install

To check that you have a successful install, run:

```
ipsec verify
```

You should see at least:

```
Checking your system to see if IPsec got installed and started correctly
Version check and ipsec on-path                                     [OK]
Checking for KLIPS support in kernel                               [OK]
Checking for RSA private key (/etc/ipsec.secrets)                  [OK]
Checking that pluto is running                                     [OK]
```

If any of these first four checks fails, see our [troubleshooting guide](#).

## Making FreeS/WAN play well with others

There are at least a couple of things on your system that might interfere with FreeS/WAN, and now's a good time to check these:

- Firewalling. You need to allow UDP 500 through your firewall, plus ESP (protocol 50) and AH (protocol 51). For more information, see our updated firewalls document (coming soon).
- Network address translation. Do not NAT the packets you will be tunneling.

## Configure for your needs

You'll need to configure FreeS/WAN for your local site. Have a look at our [opportunism quickstart guide](#) to see if that easy method is right for your needs. Or, see how to [configure a network-to-network or Road Warrior style VPN](#).

# How to configure FreeS/WAN

This page will teach you how to configure a simple network-to-network link or a Road Warrior connection between two Linux FreeS/WAN boxes.

See also these related documents:

- our [quickstart](#) guide to [opportunistic encryption](#)
- our guide to configuration with [policy groups](#)
- our [advanced configuration](#) document

The network-to-network setup allows you to connect two office networks into one Virtual Private Network, while the Road Warrior connection secures a laptop's telecommute to work. Our examples also show the basic procedure on the Linux FreeS/WAN side where another IPsec peer is in play.

Shortcut to [net-to-net](#).

Shortcut to [Road Warrior](#).

## Requirements

To configure the network-to-network connection you must have:

- two Linux gateways with static IPs
- a network behind each gate. Networks must have non-overlapping IP ranges.
- Linux FreeS/WAN [installed](#) on both gateways
- [tcpdump](#) on the local gate, to test the connection

For the Road Warrior you need:

- one Linux box with a static IP
- a Linux laptop with a dynamic IP
- Linux FreeS/WAN installed on both
- for testing, [tcpdump](#) on your gateway or laptop

If both IPs are dynamic, your situation is a bit trickier. Your best bet is a variation on the [Road Warrior](#), as described in [this mailing list message](#).

## Net-to-Net connection

### Gather information

For each gateway, compile the following information:

- gateway IP
- IP range of the subnet you will be protecting. This doesn't have to be your whole physical subnet.
- a name by which that gateway can identify itself for IPsec negotiations. Its form is a Fully Qualified Domain Name preceded by an @ sign, ie. @xy.example.com.  
It does not need to be within a domain that you own. It can be a made-up name.

### Get your lefttrsasigkey

On your local Linux FreeS/WAN gateway, print your IPsec public key:

```
ipsec showhostkey --left
```

The output should look like this (with the key shortened for easy reading):

```
# RSA 2048 bits   xy.example.com   Fri Apr 26 15:01:41 2002
leftrsasigkey=0sAQOnwiBPt...
```

Don't have a key? Use *ipsec newhostkey* to create one.

### ...and your righttrsasigkey

Get a console on the remote side:

```
ssh2 ab.example.com
```

In that window, type:

```
ipsec showhostkey --right
```

You'll see something like:

```
# RSA 2192 bits   ab.example.com   Thu May 16 15:26:20 2002
righttrsasigkey=0sAQOqH55O...
```

### Edit */etc/ipsec.conf*

Back on the local gate, copy our template to */etc/ipsec.conf*. (on Mandrake, */etc/freeswan/ipsec.conf*). Substitute the information you've gathered for our example data.

```
conn net-to-net
    left=192.0.2.2                # Local vitals
    leftsubnet=192.0.2.128/29     #
    leftid=@xy.example.com        #
    lefttrsasigkey=0s1LgR7/oUM... #
    leftnexthop=%defaultroute     # correct in many situations
    right=192.0.2.9              # Remote vitals
    rightsubnet=10.0.0.0/24       #
    rightid=@ab.example.com       #
    righttrsasigkey=0sAQOqH55O... #
    rightnexthop=%defaultroute    # correct in many situations
    auto=add                      # authorizes but doesn't start this
                                # connection at startup
```

"Left" and "right" should represent the machines that have FreeS/WAN installed on them, and "leftsubnet" and "rightsubnet" machines that are being protected. /32 is assumed for left/right and left/rightsubnet parameters.

Copy *conn net-to-net* to the remote-side */etc/ipsec.conf*. If you've made no other modifications to either *ipsec.conf*, simply:

```
scp2 ipsec.conf root@ab.example.com:/etc/ipsec.conf
```

### Start your connection

Locally, type:

```
ipsec auto --up net-to-net
```

You should see:

```
104 "net-net" #223: STATE_MAIN_I1: initiate
106 "net-net" #223: STATE_MAIN_I2: sent MI2, expecting MR2
108 "net-net" #223: STATE_MAIN_I3: sent MI3, expecting MR3
004 "net-net" #223: STATE_MAIN_I4: ISAKMP SA established
112 "net-net" #224: STATE_QUICK_I1: initiate
004 "net-net" #224: STATE_QUICK_I2: sent QI2, IPsec SA established
```

The important thing is *IPsec SA established*. If you're unsuccessful, see our [troubleshooting tips](#).

### Do not MASQ or NAT packets to be tunneled

If you are using [IP masquerade](#) or [Network Address Translation \(NAT\)](#) on either gateway, you must now exempt the packets you wish to tunnel from this treatment. For example, if you have a rule like:

```
iptables -t nat -A POSTROUTING -o eth0 -s 10.0.0.0/24 -j MASQUERADE
```

change it to something like:

```
iptables -t nat -A POSTROUTING -o eth0 -s 10.0.0.0/24 -d \! 192.0.2.128/29 -j MASQUERADE
```

This may be necessary on both gateways.

### Test your connection

Sit at one of your local subnet nodes (not the gateway), and ping a subnet node on the other (again, not the gateway).

```
ping fileserver.toledo.example.com
```

While still pinging, go to the local gateway and snoop your outgoing interface, for example:

```
tcpdump -i ppp0
```

You want to see ESP (Encapsulating Security Payload) packets moving **back and forth** between the two gateways at the same frequency as your pings:

```
19:16:32.046220 192.0.2.2 > 192.0.2.9: ESP(spi=0x3be6c4dc,seq=0x3)
19:16:32.085630 192.0.2.9 > 192.0.2.2: ESP(spi=0x5fdd1cf8,seq=0x6)
```

If you see this, congratulations are in order! You have a tunnel which will protect any IP data from one subnet to the other, as it passes between the two gates. If not, go and [troubleshoot](#).

Note: your new tunnel protects only net–net traffic, not gateway–gateway, or gateway–subnet. If you need this (for example, if machines on one net need to securely contact a fileserver on the IPsec gateway), you'll need to create extra connections.

## Finishing touches

Now that your connection works, name it something sensible, like:

```
conn winstonnet-toledonet
```

To have the tunnel come up on–boot, replace

```
auto=add
```

with:

```
auto=start
```

Copy these changes to the other side, for example:

```
scp2 ipsec.conf root@ab.example.com:/etc/ipsec.conf
```

Enjoy!

## Road Warrior Configuration

### Gather information

You'll need to know:

- the gateway's static IP
- the IP range of the subnet behind that gateway
- a name by which each side can identify itself for IPsec negotiations. Its form is a Fully Qualified Domain Name preceded by an @ sign, ie. @road.example.com.  
It does not need to be within a domain that you own. It can be a made–up name.

### Get your lefttrsasigkey...

On your laptop, print your IPsec public key:

```
ipsec showhostkey --left
```

The output should look like this (with the key shortened for easy reading):

```
# RSA 2192 bits   road.example.com   Sun Jun  9 02:45:02 2002
leftrsasigkey=0sAQPIP9uI...
```

Don't have a key? See [these instructions](#).

**...and your rightsrasigkey**

Get a console on the gateway:

```
ssh2 xy.example.com
```

View the gateway's public key with:

```
ipsec showhostkey --right
```

This will yield something like

```
# RSA 2048 bits xy.example.com Fri Apr 26 15:01:41 2002
rightsrasigkey=0sAQOnwiBPt...
```

**Customize */etc/ipsec.conf***

On your laptop, copy this template to */etc/ipsec.conf*. (on Mandrake, */etc/freeswan/ipsec.conf*). Substitute the information you've gathered for our example data.

```
conn road
    left=%defaulttroute           # Picks up our dynamic IP
    leftnexthop=%defaulttroute    #
    leftid=@road.example.com      # Local information
    leftsrasigkey=0sAQPIP9uI...   #
    right=192.0.2.10              # Remote information
    rightsubnet=10.0.0.0/24       #
    rightid=@xy.example.com       #
    rightsrasigkey=0sAQOnwiBPt... #
    auto=add                      # authorizes but doesn't start this
                                # connection at startup
```

The template for the gateway is different. Notice how it reverses *left* and *right*, in keeping with our convention that **Left** is **Local**, **Right** **Remote**. Be sure to switch your rsasigkeys in keeping with this.

```
ssh2 xy.example.com
vi /etc/ipsec.conf
```

and add:

```
conn road
    left=192.0.2.2                # Gateway's information
    leftid=@xy.example.com        #
    leftsubnet=192.0.2.128/29     #
    leftsrasigkey=0sAQOnwiBPt...  #
    rightnexthop=%defaulttroute   # correct in many situations
    right=%any                    # Wildcard: we don't know the laptop's IP
    rightid=@road.example.com     #
    rightsrasigkey=0sAQPIP9uI...  #
    auto=add                      # authorizes but doesn't start this
                                # connection at startup
```

**Start your connection**

You must start the connection from the Road Warrior side. On your laptop, type:

## Introduction to FreeS/WAN

```
ipsec auto --start net-to-net
```

You should see:

```
104 "net-net" #223: STATE_MAIN_I1: initiate
106 "road" #301: STATE_MAIN_I2: sent MI2, expecting MR2
108 "road" #301: STATE_MAIN_I3: sent MI3, expecting MR3
004 "road" #301: STATE_MAIN_I4: ISAKMP SA established
112 "road" #302: STATE_QUICK_I1: initiate
004 "road" #302: STATE_QUICK_I2: sent QI2, IPsec SA established
```

Look for *IPsec SA established*. If you're unsuccessful, see our [troubleshooting tips](#).

## Do not MASQ or NAT packets to be tunneled

If you are using [IP masquerade](#) or [Network Address Translation \(NAT\)](#) on either gateway, you must now exempt the packets you wish to tunnel from this treatment. For example, if you have a rule like:

```
iptables -t nat -A POSTROUTING -o eth0 -s 10.0.0.0/24 -j MASQUERADE
```

change it to something like:

```
iptables -t nat -A POSTROUTING -o eth0 -s 10.0.0.0/24 -d \! 192.0.2.128/29 -j MASQUERADE
```

## Test your connection

From your laptop, ping a subnet node behind the remote gateway. Do not choose the gateway itself for this test.

```
ping ns.winston.example.com
```

Snoop the packets exiting the laptop, with a command like:

```
tcpdump -i wlan0
```

You have success if you see (Encapsulating Security Payload) packets travelling **in both directions**:

```
19:16:32.046220 192.0.2.2 > 192.0.2.9: ESP(spi=0x3be6c4dc,seq=0x3)
19:16:32.085630 192.0.2.9 > 192.0.2.2: ESP(spi=0x5fdd1cf8,seq=0x6)
```

If you do, great! Traffic between your Road Warrior and the net behind your gateway is protected. If not, see our [troubleshooting hints](#).

Your new tunnel protects only traffic addressed to the net, not to the IPsec gateway itself. If you need the latter, you'll want to make an [extra tunnel](#).

## Finishing touches

On both ends, name your connection wisely, like:

```
conn mike-to-office
```

**On the laptop only**, replace

```
auto=add
```

with:

```
auto=start
```

so that you'll be connected on–boot.

Happy telecommuting!

## Multiple Road Warriors

If you're using RSA keys, as we did in this example, you can add as many Road Warriors as you like. The left/rightid parameter lets Linux FreeS/WAN distinguish between multiple Road Warrior peers, each with its own public key.

The situation is different for shared secrets (PSK). During a PSK negotiation, ID information is not available at the time Pluto is trying to determine which secret to use, so, effectively, you can only define one Roadwarrior connection. All your PSK road warriors must therefore share one secret.

## What next?

Using the principles illustrated here, you can try variations such as:

- a telecommuter with a static IP
- a road warrior with a subnet behind it

Or, look at some of our [more complex configuration examples](#).

# Linux FreeS/WAN background

This section discusses a number of issues which have three things in common:

- They are not specifically FreeS/WAN problems
- You may have to understand them to get FreeS/WAN working right
- They are not simple questions

Grouping them here lets us provide the explanations some users will need without unduly complicating the main text.

The explanations here are intended to be adequate for FreeS/WAN purposes (please comment to the [users mailing list](#) if you don't find them so), but they are not trying to be complete or definitive. If you need more information, see the references provided in each section.

## Some DNS background

[Opportunistic encryption](#) requires that the gateway systems be able to fetch public keys, and other IPsec-related information, from each other's DNS (Domain Name Service) records.

[DNS](#) is a distributed database that maps names to IP addresses and vice versa.

Much good reference material is available for DNS, including:

- the [DNS HowTo](#)
- the standard [DNS reference](#) book
- [Linux Network Administrator's Guide](#)
- [BIND overview](#)
- [BIND 9 Administrator's Reference](#)

We give only a brief overview here, intended to help you use DNS for FreeS/WAN purposes.

## Forward and reverse maps

Although the implementation is distributed, it is often useful to speak of DNS as if it were just two enormous tables:

- the forward map: look up a name, get an IP address
- the reverse map: look up an IP address, get a name

Both maps can optionally contain additional data. For opportunistic encryption, we insert the data need for IPsec authentication.

A system named gateway.example.com with IP address 10.20.30.40 should have at least two DNS records, one in each map:

*gateway.example.com. IN A 10.20.30.40*  
used to look up the name and get an IP address  
*40.30.20.10.in-addr.arpa. IN PTR gateway.example.com.*

used for reverse lookups, looking up an address to get the associated name. Notice that the digits here are in reverse order; the actual address is 10.20.30.40 but we use 40.30.20.10 here.

## Hierarchy and delegation

For both maps there is a hierarchy of DNS servers and a system of delegating authority so that, for example:

- the DNS administrator for example.com can create entries of the form *name*.example.com
- the example.com admin cannot create an entry for counterexample.com; only someone with authority for .com can do that
- an admin might have authority for 20.10.in-addr.arpa.
- in either map, authority can be delegated
  - ◆ the example.com admin could give you authority for westcoast.example.com
  - ◆ the 20.10.in-addr.arpa admin could give you authority for 30.20.10.in-addr.arpa

DNS zones are the units of delegation. There is a hierarchy of zones.

## Syntax of DNS records

Returning to the example records:

```
gateway.example.com. IN A 10.20.30.40
40.30.20.10.in-addr.arpa. IN PTR gateway.example.com.
```

some syntactic details are:

- the IN indicates that these records are for *Internet* addresses
- The final periods in '.com.' and '.arpa.' are required. They indicate the root of the domain name system.

The capitalised strings after IN indicate the type of record. Possible types include:

- *Address*, for forward lookups
- *PoinTeR*, for reverse lookups
- *Canonical NAME*, records to support aliasing, multiple names for one address
- *Mail eXchange*, used in mail routing
- *SIG*nature, used in secure DNS
- *KEY*, used in secure DNS
- *TeXT*, a multi-purpose record type

To set up for opportunistic encryption, you add some TXT records to your DNS data. Details are in our [quickstart](#) document.

## Cacheing, TTL and propagation delay

DNS information is extensively cached. With no caching, a lookup by your system of "www.freeswan.org" might involve:

- your system asks your nameserver for "www.freeswan.org"
- local nameserver asks root server about ".org", gets reply

## Introduction to FreeS/WAN

- local nameserver asks .org nameserver about "freeswan.org", gets reply
- local nameserver asks freeswan.org nameserver about "www.freeswan.org", gets reply

However, this can be a bit inefficient. For example, if you are in the Phillipines, the closest a root server is in Japan. That might send you to a .org server in the US, and then to freeswan.org in Holland. If everyone did all those lookups every time they clicked on a web link, the net would grind to a halt.

Nameservers therefore cache information they look up. When you click on another link at www.freeswan.org, your local nameserver has the IP address for that server in its cache, and no further lookups are required.

Intermediate results are also cached. If you next go to lists.freeswan.org, your nameserver can just ask the freeswan.org nameserver for that address; it does not need to query the root or .org nameservers because it has a cached address for the freeswan.org zone server.

Of course, like any cacheing mechanism, this can create problems of consistency. What if the administrator for freeswan.org changes the IP address, or the authentication key, for www.freeswan.org? If you use old information from the cache, you may get it wrong. On the other hand, you cannot afford to look up fresh information every time. Nor can you expect the freeswan.org server to notify you; that isn't in the protocols.

The solution that is in the protocols is fairly simple. Cacheable records are marked with Time To Live (TTL) information. When the time expires, the caching server discards the record. The next time someone asks for it, the server fetches a fresh copy. Of course, a server may also discard records before their TTL expires if it is running out of cache space.

This implies that there will be some delay before the new version of a changed record propagates around the net. Until the TTLs on all copies of the old record expire, some users will see it because that is what is in their cache. Other users may see the new record immediately because they don't have an old one cached.

## Problems with packet fragmentation

It seems, from mailing list reports, to be moderately common for problems to crop up in which small packets pass through the IPsec tunnels just fine but larger packets fail.

These problems are caused by various devices along the way mis-handling either packet fragments or path MTU discovery.

IPsec makes packets larger by adding an ESP or AH header. This can tickle assorted bugs in fragment handling in routers and firewalls, or in path MTU discovery mechanisms, and cause a variety of symptoms which are both annoying and, often, quite hard to diagnose.

An explanation from project technical lead Henry Spencer:

The problem is IP fragmentation; more precisely, the problem is that the second, third, etc. fragments of an IP packet are often difficult for filtering mechanisms to classify.

Routers cannot rely on reassembling the packet, or remembering what was in earlier fragments, because the fragments may be out of order or may even follow different routes. So any general, worst-case filtering decision pretty much has to be made on each fragment independently. (If the router knows that it is the only route to the destination, so all fragments \*must\* pass through it, reassembly would be possible... but most routers

## Introduction to FreeS/WAN

don't want to bother with the complications of that.)

All fragments carry roughly the original IP header, but any higher-level header is (for IP purposes) just the first part of the packet data... so only the first fragment carries that. So, for example, on examining the second fragment of a TCP packet, you could tell that it's TCP, but not what port number it is destined for -- that information is in the TCP header, which appears in the first fragment only.

The result of this classification difficulty is that stupid routers and over-paranoid firewalls may just throw fragments away. To get through them, you must reduce your MTU enough that fragmentation will not occur. (In some cases, they might be willing to attempt reassembly, but have very limited resources to devote to it, meaning that packets must be small and fragments few in number, leading to the same conclusion: smaller MTU.)

In addition to the problem Henry describes, you may also have trouble with path MTU discovery.

By default, FreeS/WAN uses a large MTU for the ipsec device. This avoids some problems, but may complicate others. Here's an explanation from Claudia:

Here are a couple of pieces of background information. Apologies if you have seen these already. An excerpt from one of my old posts:

An MTU of 16260 on ipsec0 is usual. The IPSec device defaults to this high MTU so that it does not fragment incoming packets before encryption and encapsulation. If after IPSec processing packets are larger than 1500, [ie. the mtu of eth0] then eth0 will fragment them.

Adding IPSec headers adds a certain number of bytes to each packet. The MTU of the IPSec interface refers to the maximum size of the packet before the IPSec headers are added. In some cases, people find it helpful to set ipsec0's MTU to 1500-(IPSec header size), which IIRC is about 1430.

That way, the resulting encapsulated packets don't exceed 1500. On most networks, packets less than 1500 will not need to be fragmented.

and... (from Henry Spencer)

The way it \*ought\* to work is that the MTU advertised by the ipsecN interface should be that of the underlying hardware interface, less a pinch for the extra headers needed.

Unfortunately, in certain situations this breaks many applications. There is a widespread implicit assumption that the smallest MTUs are at the ends of paths, not in the middle, and another that MTUs are never less than 1500. A lot of code is unprepared to handle paths where there is an "interior minimum" in the MTU, especially when it's less than 1500. So we advertise a big MTU and just let the resulting big packets fragment.

This usually works, but we do get bitten in cases where some intermediate point can't handle all that fragmentation. We can't win on this one.

The MTU can be changed with an *overridemtu=* statement in the *config setup* section of ipsec.conf.5.

For a discussion of MTU issues and some possible solutions using Linux advanced routing facilities, see the Linux 2.4 Advanced Routing HOWTO. For a discussion of MTU and NAT (Network Address Translation), see James Carter's MTU notes.

## Network address translation (NAT)

Network Address *T*ranslation is a service provided by some gateway machines. Calling it NAPT (adding the word *P*ort) would be more precise, but we will follow the widespread usage.

A gateway doing NAT rewrites the headers of packets it is forwarding, changing one or more of:

- source address
- source port
- destination address
- destination port

On Linux 2.4, NAT services are provided by the [netfilter\(8\)](#) firewall code. There are several [Netfilter HowTos](#) including one on NAT.

For older versions of Linux, this was referred to as "IP masquerade" and different tools were used. See this [resource page](#).

Putting an IPsec gateway behind a NAT gateway is not recommended. See our [firewalls document](#).

### NAT to non-routable addresses

The most common application of NAT uses private [non-routable](#) addresses.

Often a home or small office network will have:

- one connection to the Internet
- one assigned publicly visible IP address
- several machines that all need access to the net

Of course this poses a problem since several machines cannot use one address. The best solution might be to obtain more addresses, but often this is impractical or uneconomical.

A common solution is to have:

- [non-routable](#) addresses on the local network
- the gateway machine doing NAT
- all packets going outside the LAN rewritten to have the gateway as their source address

The client machines are set up with reserved [non-routable](#) IP addresses defined in RFC 1918. The masquerading gateway, the machine with the actual link to the Internet, rewrites packet headers so that all packets going onto the Internet appear to come from one IP address, that of its Internet interface. It then gets all the replies, does some table lookups and more header rewriting, and delivers the replies to the appropriate client machines.

As far as anyone else on the Internet is concerned, the systems behind the gateway are completely hidden. Only one machine with one IP address is visible.

For IPsec on such a gateway, you can entirely ignore the NAT in:

- [ipsec.conf\(5\)](#)
- firewall rules affecting your Internet-side interface

Those can be set up exactly as they would be if your gateway had no other systems behind it.

You do, however, have to take account of the NAT in firewall rules which affect packet forwarding.

### **NAT to routable addresses**

NAT to routable addresses is also possible, but is less common and may make for rather tricky routing problems. We will not discuss it here. See the [Netfilter HowTos](#).

# FreeS/WAN script examples

This file is intended to hold a collection of user-written example scripts or configuration files for use with FreeS/WAN.

So far it has only one entry.

## Poltorak's Firewall script

From: Poltorak Serguei <poltorak@dataforce.net>  
Subject: [Users] Using FreeS/WAN  
Date: Tue, 16 Oct 2001

Hello.

I'm using FreeS/WAN IPsec for half a year. I learned a lot of things about it and I think it would be interesting for someone to see the result of my experiments and usage of FreeS/WAN. If you find a mistake in this file, please e-mail me. And excuse me for my english... I'm learning.. :)

I'll talk about vary simple configuration:

addresses prefix = 192.168

```
lan1          sgw1      .0.0/24 (Internet)      sgw2          lan2
.1.0/24---[ .1.1 ; .0.1 ]===== [ .0.10 ; . 2.10 ]---.2.0/24
```

We need to let lan1 see lan2 across Internet like it is behind sgw1. The same for lan2. And we need to do IPX bridge for Novel Clients and NDS synchronization.

my config:

```
----- ipsec.conf -----
conn lan1-lan2
    type=tunnel
    compress=yes
    #-----
    left=192.168.0.1
    leftsubnet=192.168.1.0/24
    #-----
    right=192.168.0.10
    rightsubnet=192.168.2.0/24
    #-----
    auth=esp
    authby=secret
----- end of ipsec.conf -----
```

```
ping .2.x from .1.y    (y != 1)
It works?? Fine. Let's continue...
```

Why y != 1 ?? Because kernel of sgw1 have 2 IP addresses and it will choose the first IP (which is used to go to Internet) .0.1 and the packet won't go through IPsec tunnel :( But if do ping on .1.1 kernel will respond from that address (.1.1) and the packet will be tunneled. The same problem occurred then .2.x sends a packet to .1.2 which is down at the moment. What happens? .1.1 sends ARP requesting .1.2... after 3 tries it send to .2.x an destunreach, but from his "natural" IP or .0.1 . So the error message won't be delivered! It's a big problem...

## Introduction to FreeS/WAN

Resolution... One can manipulate with ipsec0 or ipsec0:0 to solve the problem (if ipsec0 has .1.1 kernel will send packets correctly), but there are powerful and elegant iproute2 :) We simply need to change source address of packet that goes to other secure lan. This is done with

```
ip route replace 192.168.2.0/24 via 192.168.0.10 dev ipsec0 src 192.168.1.1
```

Cool!! Now it works!!

The second step. We want install firewall on sgw1 and sgw2. Encryption of traffic without security isn't a good idea. I don't use {left|right}firewall, because I'm running firewall from init scripts.

We want IPsec data between lan1-lan2, some ICMP errors (destination unreachable, TTL exceeded, parameter problem and source quench), replying on pings from both lans and Internet, ipxtunnel data for IPX and of course SSH between sgw1 and sgw2 and from/to one specified host.

I'm using ipchains. With iptables there are some changes.

```
----- rc.firewall -----
#!/bin/sh
#
# Firewall for IPsec lan1-lan2
#

IPC=/sbin/ipchains
ANY=0.0.0.0/0

# left
SGW1_EXT=192.168.0.1
SGW1_INT=192.168.1.1
LAN1=192.168.1.0/24

# right
SGW2_EXT=192.168.0.10
SGW2_INT=192.168.2.10
LAN2=192.168.2.0/24

# SSH from and to this host
SSH_PEER_HOST=_SOME_HOST_

# this is for left. exchange these values for right.
MY_EXT=$SGW1_EXT
MY_INT=$SGW1_INT
PEER_EXT=$SGW2_EXT
PEER_INT=$SGW2_INT
INT_IF=eth1
EXT_IF=eth0
IPSEC_IF=ipsec0
MY_LAN=$LAN1
PEER_LAN=$LAN2

$IPC -F
$IPC -P input DENY
$IPC -P forward DENY
$IPC -P output DENY

# Loopback traffic
$IPC -A input -i lo -j ACCEPT
$IPC -A output -i lo -j ACCEPT
```

## Introduction to FreeS/WAN

```
# for IPsec SGW1-SGW2
## IKE
$IPC -A input -p udp -s $PEER_EXT 500 -d $MY_EXT 500 -i $EXT_IF -j ACCEPT
$IPC -A output -p udp -s $MY_EXT 500 -d $PEER_EXT 500 -i $EXT_IF -j ACCEPT
## ESP
$IPC -A input -p 50 -s $PEER_EXT -d $MY_EXT -i $EXT_IF -j ACCEPT
### we don't need this line ### $IPC -A output -p 50 -s $MY_EXT -d $PEER_EXT -i $EXT_IF -j ACCEPT
## forward LAN1-LAN2
$IPC -A forward -s $MY_LAN -d $PEER_LAN -i $IPSEC_IF -j ACCEPT
$IPC -A forward -s $PEER_LAN -d $MY_LAN -i $INT_IF -j ACCEPT
$IPC -A output -s $PEER_LAN -d $MY_LAN -i $INT_IF -j ACCEPT
$IPC -A input -s $PEER_LAN -d $MY_LAN -i $IPSEC_IF -j ACCEPT
$IPC -A input -s $MY_LAN -d $PEER_LAN -i $INT_IF -j ACCEPT
$IPC -A output -s $MY_LAN -d $PEER_LAN -i $IPSEC_IF -j ACCEPT

# ICMP
#
## Dest unreachable
### from/to Internet
$IPC -A input -p icmp --icmp-type destination-unreachable -s $ANY -d $MY_EXT -i $EXT_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type destination-unreachable -s $MY_EXT -d $ANY -i $EXT_IF -j ACCEPT
### from/to Lan
$IPC -A input -p icmp --icmp-type destination-unreachable -s $ANY -d $MY_INT -i $INT_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type destination-unreachable -s $MY_INT -d $ANY -i $INT_IF -j ACCEPT
### from/to Peer Lan
$IPC -A input -p icmp --icmp-type destination-unreachable -s $ANY -d $MY_INT -i $IPSEC_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type destination-unreachable -s $MY_INT -d $ANY -i $IPSEC_IF -j ACCEPT
#
## Source quench
### from/to Internet
$IPC -A input -p icmp --icmp-type source-quench -s $ANY -d $MY_EXT -i $EXT_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type source-quench -s $MY_EXT -d $ANY -i $EXT_IF -j ACCEPT
### from/to Lan
$IPC -A input -p icmp --icmp-type source-quench -s $ANY -d $MY_INT -i $INT_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type source-quench -s $MY_INT -d $ANY -i $INT_IF -j ACCEPT
### from/to Peer Lan
$IPC -A input -p icmp --icmp-type source-quench -s $ANY -d $MY_INT -i $IPSEC_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type source-quench -s $MY_INT -d $ANY -i $IPSEC_IF -j ACCEPT
#
## Parameter problem
### from/to Internet
$IPC -A input -p icmp --icmp-type parameter-problem -s $ANY -d $MY_EXT -i $EXT_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type parameter-problem -s $MY_EXT -d $ANY -i $EXT_IF -j ACCEPT
### from/to Lan
$IPC -A input -p icmp --icmp-type parameter-problem -s $ANY -d $MY_INT -i $INT_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type parameter-problem -s $MY_INT -d $ANY -i $INT_IF -j ACCEPT
### from/to Peer Lan
$IPC -A input -p icmp --icmp-type parameter-problem -s $ANY -d $MY_INT -i $IPSEC_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type parameter-problem -s $MY_INT -d $ANY -i $IPSEC_IF -j ACCEPT
#
## Time To Live exceeded
### from/to Internet
$IPC -A input -p icmp --icmp-type time-exceeded -s $ANY -d $MY_EXT -i $EXT_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type time-exceeded -s $MY_EXT -d $ANY -i $EXT_IF -j ACCEPT
### to Lan
$IPC -A input -p icmp --icmp-type time-exceeded -s $ANY -d $MY_INT -i $INT_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type time-exceeded -s $MY_INT -d $ANY -i $INT_IF -j ACCEPT
### to Peer Lan
$IPC -A input -p icmp --icmp-type time-exceeded -s $ANY -d $MY_INT -i $IPSEC_IF -j ACCEPT
$IPC -A output -p icmp --icmp-type time-exceeded -s $MY_INT -d $ANY -i $IPSEC_IF -j ACCEPT
```

## Introduction to FreeS/WAN

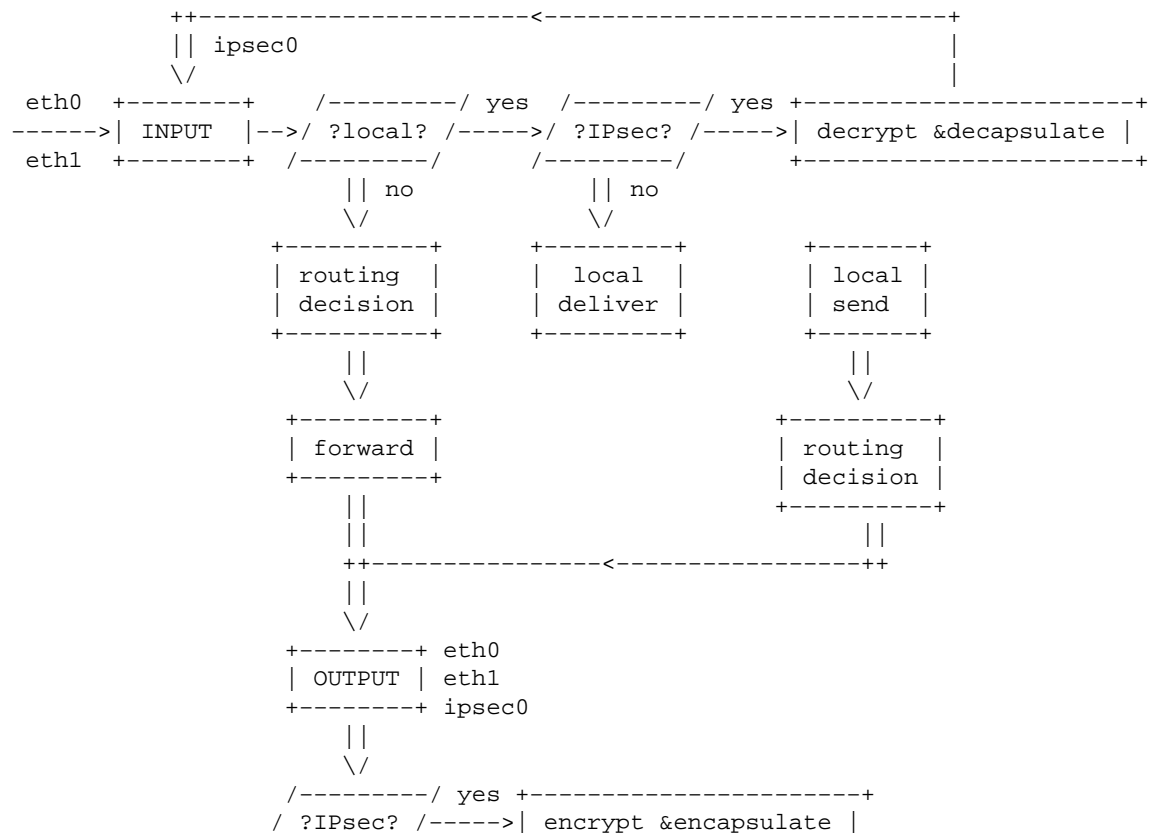
```
# ICMP PINGS
## from Internet
$IPC -A input -p icmp -s $ANY -d $MY_EXT --icmp-type echo-request -i $EXT_IF -j ACCEPT
$IPC -A output -p icmp -s $MY_EXT -d $ANY --icmp-type echo-reply -i $EXT_IF -j ACCEPT
## from LAN
$IPC -A input -p icmp -s $ANY -d $MY_INT --icmp-type echo-request -i $INT_IF -j ACCEPT
$IPC -A output -p icmp -s $MY_INT -d $ANY --icmp-type echo-reply -i $INT_IF -j ACCEPT
## from Peer LAN
$IPC -A input -p icmp -s $ANY -d $MY_INT --icmp-type echo-request -i $IPSEC_IF -j ACCEPT
$IPC -A output -p icmp -s $MY_INT -d $ANY --icmp-type echo-reply -i $IPSEC_IF -j ACCEPT

# SSH
## from SSH_PEER_HOST
$IPC -A input -p tcp -s $SSH_PEER_HOST -d $MY_EXT 22 -i $EXT_IF -j ACCEPT
$IPC -A output -p tcp \! -y -s $MY_EXT 22 -d $SSH_PEER_HOST -i $EXT_IF -j ACCEPT
## to SSH_PEER_HOST
$IPC -A input -p tcp \! -y -s $SSH_PEER_HOST 22 -d $MY_EXT -i $EXT_IF -j ACCEPT
$IPC -A output -p tcp -s $MY_EXT -d $SSH_PEER_HOST 22 -i $EXT_IF -j ACCEPT
## from PEER
$IPC -A input -p tcp -s $PEER_EXT -d $MY_EXT 22 -i $EXT_IF -j ACCEPT
$IPC -A output -p tcp \! -y -s $MY_EXT 22 -d $PEER_EXT -i $EXT_IF -j ACCEPT
## to PEER
$IPC -A input -p tcp \! -y -s $PEER_EXT 22 -d $MY_EXT -i $EXT_IF -j ACCEPT
$IPC -A output -p tcp -s $MY_EXT -d $PEER_EXT 22 -i $EXT_IF -j ACCEPT

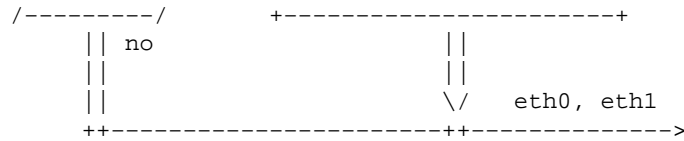
# ipxtunnel
$IPC -A input -p udp -s $PEER_INT 2005 -d $MY_INT 2005 -i $IPSEC_IF -j ACCEPT
$IPC -A output -p udp -s $MY_INT 2005 -d $PEER_INT 2005 -i $IPSEC_IF -j ACCEPT
```

----- end of rc.firewall -----

To understand this we need to look on this scheme:



## Introduction to FreeS/WAN



This explain how a packet traverse TCP/IP stack in IPsec capable kernel.

FIX ME, please, if there are any errors

Test the new firewall now.

Now about IPX. I tried 3 programs for tunneling IPX: tipxd, SIB and ipxtunnel

tipxd didn't send packets.. :(  
SIB and ipxtunnel worked fine :)  
With ipxtunnel there was a little problem. In sources there are an error.

```

----- in main.c -----
<      bytes += p.len;
---
>      bytes += len;
-----

```

After this FIX everything goes right...

```

----- /etc/ipxtunnel.conf -----
port      2005
remote    192.168.101.97      2005
interface eth1
----- end of /etc/ipxtunnel.conf -----

```

I use IPX tunnel between .1.1 and .2.10 so we don't need to encrypt nor authenticate encapsulated IPX packets, it is done with IPsec.

If you don't wont to use iproute2 to change source IP you need to use SIB (it is able to bind local address) or establish tunnel between .0.1 and .0.10 (external IPs, you need to do encryption in the program, but it isn't strong).

For now I'm using ipxtunnel.

I think that's all for the moment. If there are any error, please e-mail me: poltorak@df.ru . It would be cool if someone puts the scheme of TCP/IP in kernel and firewall example on FreeS/WAN's manual pages.

PoltoS

# How to configure to use "make check"

## What is "make check"

"make check" is a target in the top level makefile. It takes care of running a number of unit and system tests to confirm that FreeSWAN has been compiled correctly, and that no new bugs have been introduced.

As FreeSWAN contains both kernel and userspace components, doing testing of FreeSWAN requires that the kernel be simulated. This is typically difficult to do as a kernel requires that it be run on bare hardware. A technology has emerged that makes this simpler. This is User Mode Linux.

User-Mode Linux is a way to build a Linux kernel such that it can run as a process under another Linux (or in the future other) kernel. Presently, this can only be done for 2.4 guest kernels. The host kernel can be 2.2 or 2.4.

"make check" expects to be able to build User-Mode Linux kernels with FreeSWAN included. To do this it needs to have some files downloaded and extracted prior to running "make check". This is described in the UML testing document.

After having run the example in the UML testing document and successfully brought up the four machine combination, you are ready to use "make check"

## Running "make check"

"make check" works by walking the FreeSWAN source tree invoking the "check" target at each node. At present there are tests defined only for the `klips` directory. These tests will use the UML infrastructure to test out pieces of the `klips` code.

The results of the tests can be recorded. If the environment variable `$REGRESSRESULTS` is non-null, then the results of each test will be recorded. This can be used as part of a nightly regression testing system, see Nightly testing for more details.

"make check" otherwise prints a minimal amount of output for each test, and indicates pass/fail status of each test as they are run. Failed tests do not cause failure of the target in the form of exit codes.

# How to write a "make check" test

## Structure of a test

Each test consists of a set of directories under `testing/`. There are directories for `klips`, `pluto`, `packaging` and `libraries`. Each directory has a list of tests to run is stored in a file called `TESTLIST` in that directory. e.g. `testing/klips/TESTLIST`.

## The TESTLIST

This isn't actually a shell script. It just looks like one. Some tools other than `/bin/sh` process it. Lines that start with `#` are comments.

```
# test-kind      directory-containing-test      expectation      [PR#]
```

The first word provides the test type, detailed below.

The second word is the name of the test to run. This the directory in which the test case is to be found..

The third word may be one of:

*blank/good*

the test is believed to function, report failure

*bad*

the test is known to fail, report unexpected success

*suspended*

the test should not be run

The fourth word may be a number, which is a PR# if the test is failing.

## Test kinds

The test types are:

*skiptest*

means run no test.

*ctltest*

means run a single system without input/output.

*klipstest*

means run a single system with input/output networks

*umlplutotest*

means run a pair of systems

*umlXhost*

run an arbitrary number of systems

*suntest (TBD)*

means run a quad of east/west/sunrise/sunset

*roadtest (TBD)*

means run a trio of east-sunrise + warrior

*extrudedtest (TBD)*

means run a quad of east-sunrise + warriorsouth + park

*mkinsttest*

a test of the "make install" machinery.

*kernel\_test\_patch*

a test of the "make kernelpatch" machinery.

Tests marked (TBD) have yet to be fully defined.

Each test directory has a file in it called `testparams.sh`. This file sets a number of environment variables to define the parameters of the test.

## Common parameters

*TESTNAME*

the name of the test (repeated for checking purposes)

*TEST\_TYPE*

the type of the test (repeat of type type above)

*TESTHOST*

the name of the UML machine to run for the test, typically "east" or "west"

*TEST\_PURPOSE*

The purpose of the test is one of:

*goal*

The goal purpose is where a test is defined for code that is not yet finished. The test indicates when the goals have in fact been reached.

*regress*

This is a test to determine that a previously existing bug has been repaired. This test will initially be created to reproduce the bug in isolation, and then the bug will be fixed.

*exploit*

This is a set of packets/programs that causes a vulnerability to be exposed. It is a specific variation of the regress option.

*TEST\_GOAL\_ITEM*

in the case of a goal test, this is a reference to the requirements document

*TEST\_PROB\_REPORT*

in the case of regression test, this the problem report number from GNATS

*TEST\_EXPLOIT\_URL*

in the case of an exploit, this is a URL referencing the paper explaining the origin of the test and the origin of exploit software

*REF\_CONSOLE\_OUTPUT*

a file in the test directory that contains the sanitized console output against which to compare the output of the actual test.

*REF\_CONSOLE\_FIXUPS*

a list of scripts (found in `klips/test/fixups`) to apply to sanitize the console output of the machine under test. These are typically perl, awk or sed scripts that remove things in the kernel output that change each time the test is run and/or compiled.

*INIT\_SCRIPT*

a file of commands that is fed into the virtual machine's console in single user mode prior to starting the tests. This file will usually set up any eroute's and SADB entries that are required for the test.

Lines beginning with # are skipped. Blank lines are skipped. Otherwise, a shell prompt is waited for each time (consisting of \n#) and then the command is sent. Note that the prompt is waited for before the command and not after, so completion of the last command in the script is not required. This is often used to invoke a program to monitor the system, e.g. `ipsec pf_key`.

### *RUN\_SCRIPT*

a file of commands that is fed into the virtual machine's console in single user mode, before the packets are sent. On single machine tests, this script doesn't provide any more power than `INIT_SCRIPT`, but is implemented for consistency's sake.

### *FINAL\_SCRIPT*

a file of commands that is fed into the virtual machine's console in single user mode after the final packet is sent. Similar to `INIT_SCRIPT`, above. If not specified, then the single command "halt" is sent. If specified, then the script should end with a halt command to nicely shutdown the UML.

### *CONSOLEDIFFDEBUG*

If set to "true" then the series of console fixups (see `REF_CONSOLE_FIXUPS`) will be output after it is constructed. (It should be set to "false", or unset otherwise)

### *NETJIGDEBUG*

If set to "true" then the series of console fixups (see `REF_CONSOLE_FIXUPS`) will be output after it is constructed. (It should be set to "false", or unset otherwise)

### *NETJIGTESTDEBUG*

If set to "netjig", then the results of talking to the `uml_netjig` will be printed to stderr during the test. In addition, the jig will be invoked with `--debug`, which causes it to log its process ID, and wait 60 seconds before continuing. This can be used if you are trying to debug the `uml_netjig` program itself.

### *HOSTTESTDEBUG*

If set to "hosttest", then the results of talking to the consoles of the UMLs will be printed to stderr during the test.

### *NETJIGWAITUSER*

If set to "waituser", then the scripts will wait forever for user input before they shut the tests down. Use this if you are debugging through the kernel.

### *PACKETRATE*

A number, in milliseconds (default is 500ms) at which packets will be replayed by the netjig.

## KLIPStest paramaters

The `klipstest` function starts a program (`testing/utils/uml_netjig/uml_netjig`) to setup a bunch of I/O sockets (that simulate network interfaces). It then exports the references to these sockets to the environment and invokes (using `system()`) a given script. It waits for the script to finish.

The script invoked (`testing/utils/host-test.tcl`) is a TCL expect script that arranges to start the UML and configure it appropriately for the test. The configuration is done with the script given above for `INIT_SCRIPT`. The TCL script then forks, leaves the UML in the background and exits. `uml_netjig` continues. It then starts listening to the simulated network answering ARPs and inserting packets as appropriate.

The `klipstest` function invokes `uml_netjig` with arguments to capture output from network interface(s) and insert packets as appropriate:

### *PUB\_INPUT*

a pcap file to feed in on the public (encrypted) interface. (typically, `eth1`)

### *PRIV\_INPUT*

a pcap file to feed in on the private (plain-text) interface (typically, eth0).

### *REF\_PUB\_OUTPUT*

a text file containing tcpdump output. Packets on the public (eth1) interface are captured to a pcap file by `uml_netjig`. The `klipstest` function then uses tcpdump on the file to produce text output, which is compared to the file given.

### *REF\_PUB\_FILTER*

a program that will filter the TCPDUMP output to do further processing. Defaults to "cat".

### *REF\_PRIV\_OUTPUT*

a text file containing tcpdump output. Packets on the private (eth0) interface are captured and compared after conversion by tcpdump, as with *REFPUBOUTPUT*.

### *REF\_PRIV\_FILTER*

a program that will filter the TCPDUMP output to do further processing. Defaults to "cat".

### *EXITONEMPTY*

a flag for `uml_netjig`. It should contain "--exitonempty" if `uml_netjig` should exit when all of the input (*PUBINPUT*, *PRIVINPUT*) packets have been injected.

### *ARPREPLY*

a flag for `uml_netjig`. It should contain "--arpreply" if `uml_netjig` should reply to ARP requests. One will typically set this to avoid having to fudge the ARP cache manually.

### *TCPDUMPFLAGS*

a set of flags for the tcpdump used when converting captured output. Typical values will include "-n" to turn off DNS, and often "-E" to set the decryption key (tcpdump 3.7.1 and higher only) for ESP packets. The "-t" flag (turn off timestamps) is provided automatically

### *NETJIG\_EXTRA*

additional comments to be sent to the netjig. This may arrange to record or create additional networks, or may toggle options.

## mkinsttest paramaters

The basic concept of the `mkinsttest` test type is that it performs a "make install" to a temporary `$DESTDIR`. The resulting tree can then be examined to determine if it was done properly. The files can be uninstalled to determine if the file list was correct, or the contents of files can be examined more precisely.

### *INSTALL\_FLAGS*

If set, then an install will be done. This provides the set of flags to provide for the install. The target to be used (usually "install") must be among the flags.

### *POSTINSTALL\_SCRIPT*

If set, a script to run after initial "make install". Two arguments are provided: an absolute path to the root of the FreeSWAN src tree, and an absolute path to the temporary installation area.

### *INSTALL2\_FLAGS*

If set, a second install will be done using these flags. Similarly to *INSTALL\_FLAGS*, the target must be among the flags.

### *UNINSTALL\_FLAGS*

If set, an uninstall will be done using these flags. Similarly to *INSTALL\_FLAGS*, the target (usually "uninstall") must be among the flags.

### *REF\_FIND\_f\_l\_OUTPUT*

If set, a `find $ROOT ( -type f -or -type -l )` will be done to get a list of a real files and symlinks. The resulting file will be compared to the file listed by this option.

### *REF\_FILE\_CONTENTS*

If set, it should point to a file containing records for the form:

```
reffile    samplefile
```

one record per line. A diff between the provided reference file, and the sample file (located in the temporary installation root) will be done for each record.

## **rpm\_build\_install\_test paramaters**

The `rpm_build_install_test` type is to verify that the proper packing list is produced by "make rpm", and that the mechanisms for building the kernel modules produce consistent results.

### *RPM\_KERNEL\_SOURCE*

Point to an extracted copy of the RedHat kernel source code. Variables from the environment may be used.

### *REF\_RPM\_CONTENTS*

This is a file containing one record per line. Each record consists of a RPM name (may contain wildcards) and a filename to compare the contents to. The RPM will be located and a file list will be produced with `rpm2cpio`.

## **libtest paramaters**

The `libtest` test is for testing library routines. The library file is expected to provided an `#ifdef` by the name of *library*. The `libtest` type invokes the C compiler to compile this file, links it against `libfreeswan.a` (to resolve any other dependancies) and runs the test with the `-r` argument to invoke a regression test.

The library test case is expected to do a self-test, exiting with status code 0 if everything is okay, and with non-zero otherwise. A core dump (exit code greater than 128) is noted specifically.

Unlike other tests, there are no subdirectories required, or other parameters to set.

## **umlplutotest paramaters**

The `umlplutotest` function starts a pair of user mode line processes. This is a 2-host version of `umlXhost`. The "EAST" and "WEST" slots are defined.

## **umlXhost parameters**

The `umlXtest` function starts an arbitrary number of user mode line processes.

The script invoked (`testing/utils/Xhost-test.tcl`) is a TCL expect script that arranges to start each UML and configure it appropriately for the test. It then starts listening (using `uml_netjig`) to the simulated network answering ARPs and inserting packets as appropriate.

`umlXtest` has a series of slots, each of which should be filled by a host. The list of slots is controlled by the variable, `XHOST_LIST`. This variable should be set to a space separated list of slots. The former `umlplutotest` is now implemented as a variation of the `umlXhost` test, with `XHOST_LIST="EAST WEST"`.

For each host slot that is defined, a series of variables should be filled in, defining what configuration scripts to use for that host.

## Introduction to FreeS/WAN

The following are used to control the console input and output to the system. Where the string `${host}` is present, the host slot should be filled in. I.e. for the two host system with `XHOST_LIST="EAST WEST"`, then the variables: `EAST_INIT_SCRIPT` and `WEST_INIT_SCRIPT` will exist.

### *`${host}HOST`*

The name of the UML host which will fill this slot

### *`${host}_INIT_SCRIPT`*

a file of commands that is fed into the virtual machine's console in single user mode prior to starting the tests. This file will usually set up any eroute's and SADB entries that are required for the test.

Similar to `INIT_SCRIPT`, above.

### *`${host}_RUN_SCRIPT`*

a file of commands that is fed into the virtual machine's console in single user mode, before the packets are sent. This set of commands is run after all of the virtual machines are initialized. I.e. after `EAST_INIT_SCRIPT` **AND** `WEST_INIT_SCRIPT`. This script can therefore do things that require that all machines are properly configured.

### *`${host}_RUN2_SCRIPT`*

a file of commands that is fed into the virtual machine's console in single user mode, after the packets are sent. This set of commands is run before any of the virtual machines have been shut down. (I.e. before `EAST_FINAL_SCRIPT` **AND** `WEST_FINAL_SCRIPT`.) This script can therefore catch post-activity status reports.

### *`${host}_FINAL_SCRIPT`*

a file of commands that is fed into the virtual machine's console in single user mode after the final packet is sent. Similar to `INIT_SCRIPT`, above. If not specified, then the single command "halt" is sent. Note that when this script is run, the other virtual machines may already have been killed. If specified, then the script should end with a halt command to nicely shutdown the UML.

### *`REF_${host}_CONSOLE_OUTPUT`*

Similar to `REF_CONSOLE_OUTPUT`, above.

Some additional flags apply to all hosts:

### *`REF_CONSOLE_FIXUPS`*

a list of scripts (found in `klips/test/fixups`) to apply to sanitize the console output of the machine under test. These are typically perl, awk or sed scripts that remove things in the kernel output that change each time the test is run and/or compiled.

In addition to input to the console, the networks may have input fed to them:

### *`EAST_INPUT/WEST_INPUT`*

a pcap file to feed in on the private network side of each network. The "EAST" and "WEST" here refer to the networks, not the hosts.

### *`REF_PUB_FILTER`*

a program that will filter the `TCPDUMP` output to do further processing. Defaults to "cat".

### *`REF_EAST_FILTER/REF_WEST_FILTER`*

a program that will filter the `TCPDUMP` output to do further processing. Defaults to "cat".

<

### *`TCPDUMPFLAGS`*

a set of flags for the `tcpdump` used when converting captured output. Typical values will include "-n" to turn off DNS, and often "-E" to set the decryption key (`tcpdump` 3.7.1 and higher only) for ESP packets. The "-t" flag (turn off timestamps) is provided automatically

### *`REF_EAST_OUTPUT/REF_WEST_OUTPUT`*

a text file containing tcpdump output. Packets on the private (eth0) interface are captured and compared after conversion by tcpdump, as with *REF\_PUB\_OUTPUT*.

There are two additional environment variables that may be set on the command line:

*NETJIGVERBOSE=verbose export NETJIGVERBOSE*

If set, then the test output will be "chatty", and let you know what commands it is running, and as packets are sent. Without it set, the output is limited to success/failure messages.

*NETJIGTESTDEBUG=netjig export NETJIGTESTDEBUG*

This will enable debugging of the communication with `uml_netjig`, and turn on debugging in this utility. This does not imply *NETJIGVERBOSE*.

*HOSTTESTDEBUG=hosttest export HOSTTESTDEBUG*

This will show all interactions with the user-mode-linux consoles

## kernel\_patch\_test paramaters

The `kernel_patch_test` function takes some kernel source, copies it with `Indir`, and then applies the patch as produced by "make kernelpatch".

The following are used to control the input and output to the system:

*KERNEL\_NAME*

the kernel name, typically something like "linus" or "rh"

*KERNEL\_VERSION*

the kernel version number, as in "2.2" or "2.4".

*KERNEL\_\${KERNEL\_NAME}\_\${KERNEL\_VERSION}\_SRC*

This variable should set in the environment, probably in `~/freeswan-regress-env.sh`. Examples of this variables would be `KERNEL_LINUS2_0_SRC` or `KERNEL_RH7_3_SRC`. This variable should point to an extracted copy of the kernel source in question.

*REF\_PATCH\_OUTPUT*

a copy of the patch output to compare against

*KERNEL\_PATCH\_LEAVE\_SOURCE*

If set to a non-empty string, then the patched kernel source is not removed at the end of the test. This will typically be set in the environment while debugging.

## module\_compile paramaters

The `module_compile` test attempts to build the KLIPS module against a given set of kernel source. This is also done by the RPM tests, but in a very specific manner.

There are two variations of this test – one where the kernel either doesn't need to be configured, or is already done, and tests were there is a local configuration file.

Where the kernel doesn't need to be configured, the kernel source that is found is simply used. It may be a RedHat-style kernel, where one can cause it to configure itself via `rhconfig.h`-style definitions. Or, it may just be a kernel tree that has been configured.

If the variable `KERNEL_CONFIG_FILE` is set, then a new directory is created for the kernel source. It is populated with `Indir(1)`. The referenced file is then copied in as `.config`, and "make oldconfig" is used to configure the kernel. This resulting kernel is then used as the reference source.

## Introduction to FreeS/WAN

In all cases, the kernel source is found the same was for the kernelpatch test, i.e. via  
KERNEL\_VERSION/KERNEL\_NAME and  
KERNEL\_\${KERNEL\_NAME}\${KERNEL\_VERSION}\_SRC.

Once there is kernel source, the module is compiled using the top-level "make module" target.

The test is considered successful if an executable is found in OUTPUT/module/ipsec.o at the end of the test.

### *KERNEL\_NAME*

the kernel name, typically something like "linus" or "rh"

### *KERNEL\_VERSION*

the kernel version number, as in "2.2" or "2.4".

### *KERNEL\_\${KERNEL\_NAME}\${KERNEL\_VERSION}\_SRC*

This variable should set in the environment, probably in ~/freeswan-regress-env.sh. Examples of this variables would be KERNEL\_LINUX2\_0\_SRC or KERNEL\_RH7\_3\_SRC. This variable should point to an extracted copy of the kernel source in question.

### *KERNEL\_CONFIG\_FILE*

The configuration file for the kernel.

### *KERNEL\_PATCH\_LEAVE\_SOURCE*

If set to a non-empty string, then the configured kernel source is not removed at the end of the test.

This will typically be set in the environment while debugging.

### *MODULE\_DEF\_INCLUDE*

The include file that will be used to configure the KLIPS module, and possibly the kernel source.

# Current pitfalls

*"tcpdump dissector" not available.*

This is a non-fatal warning. If `uml_netjig` is invoked with the `-t` option, then it will attempt to use `tcpdump`'s dissector to decode each packet that it processes. The dissector is presently not available, so this option is normally turned off at compile time. The dissector library will be released with `tcpdump` version 4.0.

# User-Mode-Linux Testing guide

User mode linux is a way to compile a linux kernel such that it can run as a process in another linux system (potentially as a \*BSD or Windows process later). See <http://user-mode-linux.sourceforge.net/>

UML is a good platform for testing and experimenting with FreeS/WAN. It allows several network nodes to be simulated on a single machine. Creating, configuring, installing, monitoring, and controlling these nodes is generally easier and easier to script with UML than real hardware.

You'll need about 500Mb of disk space for a full sunrise-east-west-sunset setup. You can possibly get this down by 130Mb if you remove the sunrise/sunset kernel build. If you just want to run, then you can even remove the east/west kernel build.

Nothing need be done as super user. In a couple of steps, we note where super user is required to install commands in system-wide directories, but ~/bin could be used instead. UML seems to use a system-wide /tmp/uml directory so different users may interfere with one another. Later UMLs use ~/.uml instead, so multiple users running UML tests should not be a problem, but note that a single user running the UML tests will only be able run one set. Further, UMLs sometimes get stuck and hang around. These "zombies" (most will actually be in the "T" state in the process table) will interfere with subsequent tests.

## Preliminary Notes on BIND

As of 2003/3/1, the Light-Weight Resolver is used by pluto. This requires that BIND9 be running. It also requires that BIND9 development libraries be present in the build environment. The DNSSEC code is only truly functional in BIND9 snapshots. The library code could be 9.2.2, we believe. We are using BIND9 20021115 snapshot code from <ftp://ftp.isc.org/isc/bind9/snapshots>.

FreeS/WAN may well require a newer BIND than is on your system. Many distributions have moved to BIND9.2.2 recently due to a security advisory. BIND is five components.

1. named
2. dnssec-\*
3. client side resolver libraries
4. client side utility libraries I thought there were lib and named parts to dnssec...
5. dynamic DNS update utilities

The only piece that we need for \*building\* is #4. That's the only part that has to be on the build host. What is the difference between resolver and util libs? If you want to edit testing/baseconfigs/all/etc/bind, you'll need a snapshot version. The resolver library contains the resolver. FreeS/WAN has its own copy of that in lib/liblwres.

## Steps to Install UML for FreeS/WAN

1. Get the following files:
  - a. from <http://www.sandelman.ottawa.on.ca/freeswan/uml/> umlfreeroot-15.1.tar.gz (or highest numbered one). This is a debian potato root file system. You can use this even on a Redhat host, as it has the newer GLIBC2.2 libraries as well.
  - b. From <ftp://ftp.xs4all.nl/pub/crypto/freeswan/> a snapshot or release (1.92 or better)

## Introduction to FreeS/WAN

- c. From a <http://www.kernel.org/mirror>, the virgin 2.4.19 kernel. Please realize that we have defaults in our tree for kernel configuration. We try to track the latest UML kernels. If you use a newer kernel, you may have faults in the kernel build process. You can see what the latest that is being regularly tested by visiting [freeswan-regress-env.sh](http://freeswan-regress-env.sh).
  - d. Get <http://ftp.nl.linux.org/uml/> uml-patch-2.4.19-47.bz2 or the one associated with your kernel. As of 2003/03/05, uml-patch-2.4.19-47.bz2 works for us. ***More recent versions of the patch have not been tested by us.***
  - e. You'll probably want to visit <http://user-mode-linux.sourceforge.net> and get the UML utilities. These are not needed for the build or interactive use (but recommended). They are necessary for the regression testing procedures used by "make check". We currently use uml\_utilities\_20020212.tar.bz2.
  - f. You need tcpdump version 3.7.1 or better. This is newer than the version included in most LINUX distributions. You can check the version of an installed tcpdump with the --version flag. If you need a newer tcpdump fetch both tcpdump and libpcap source tar files from <http://www.tcpdump.org/> or a mirror.
2. Pick a suitable place, and extract the following files:
- a. 2.4.19 kernel. For instance:

```
cd /c2/kernel
tar xzvf ../download/pub/linux/kernel/v2.4/linux-2.4.19.tar.gz
```

- b. extract the umlfreeroot file

```
mkdir -p /c2/user-mode-linux/basic-root
cd /c2/user-mode-linux/basic-root
tar xzvf ../download/umlfreeroot-15.1.tar.gz
```

- c. FreeSWAN itself (or checkout "all" from CVS)

```
mkdir -p /c2/freeswan/sandbox
cd /c2/freeswan/sandbox
tar xzvf ../download/snapshot.tar.gz
```

3. If you need to build a newer tcpdump:
- ◆ Make sure you have OpenSSL installed — it is needed for cryptographic routines.
  - ◆ Unpack libpcap and tcpdump source in parallel directories (the tcpdump build procedures look for libpcap next door).
  - ◆ Change directory into the libpcap source directory and then build the library:

```
./configure
make
```

- ◆ Change into the tcpdump source directory, build tcpdump, and install it.

```
./configure
make
# Need to be superuser to install in system directories.
# Installing in ~/bin would be an alternative.
su -c "make install"
```

4. If you need the uml utilities, unpack them somewhere then build and install them:

```
cd tools
make all
# Need to be superuser to install in system directories.
```

## Introduction to FreeS/WAN

```
# Installing in ~/bin would be an alternative.
su -c "make install BIN_DIR=/usr/local/bin"
```

### 5. set up the configuration file

- ◆ `cd /c2/freeswan/sandbox/freeswan-1.97/testing/utils`
- ◆ copy `umlsetup-sample.sh` to `../.. /umlsetup.sh`: `cp umlsetup-sample.sh ../.. /umlsetup.sh`
- ◆ open up `../.. /umlsetup.sh` in your favorite editor.
- ◆ change `POOLSPACE=` to point to the place with at least 500Mb of disk. Best if it is on the same partition as the "umlfreeroot" extraction, as it will attempt to use hard links if possible to save disk space.
- ◆ Set `TESTINGROOT` if you intend to run the script outside of the sandbox/snapshot/release directory. Otherwise, it will configure itself.
- ◆ `KERNPOOL` should point to the directory with your 2.4.19 kernel tree. This tree should be unconfigured! This is the directory you used in step 2a.
- ◆ `UMLPATCH` should point at the bz2 file you downloaded at 1d. If using a kernel that already includes the patch, set this to `/dev/null`.
- ◆ `FREESWANDIR` should point at the directory where you unpacked the snapshot/release. Include the "freeswan-snap2001sep16b" or whatever in it. If you are running from CVS, then you point at the directory where `top`, `klips`, etc. are. The script will fix up the directory so that it can be used.
- ◆ `BASICROOT` should be set to the directory used in 2b, or to the directory that you created with RPMs.
- ◆ `SHAREDIR` should be set to the directory used in 2c, to `/usr/share` for Debian potato users, or to `$BASICROOT/usr/share`.

### 6. `cd $TESTINGROOT/utils` `sh make-uml.sh`

It will grind for awhile. If there are errors it will bail. If so, run it under "script" and send the output to [bugs@lists.freeswan.org](mailto:bugs@lists.freeswan.org).

### 7. You will have a bunch of stuff under `$POOLSPACE`. Open four xterms:

```
for i in sunrise sunset east west
do
    xterm -name $i -title $i -e $POOLSPACE/$i/start.sh &    done
```

8. Login as root. Password is "root" (Note, these virtual machines are networked together, but are not configured to talk to the rest of the world.)
9. verify that pluto started on east/west, run "ipsec look"
10. login to sunrise. run "ping sunset"
11. login to west. run "tcpdump -p -i eth1 -n" (tcpdump must be version 3.7.1 or newer)
12. Closing a console xterm will shut down that UML.
13. You can "make check", if you want to. It is run from `/c2/freeswan/sandbox/freeswan-1.97`.

# Debugging the kernel with GDB

With User-Mode-Linux, you can debug the kernel using GDB. See <http://user-mode-linux.sourceforge.net/debugging.html>.

Typically, one will want to address a test case for a failing situation. Running GDB from Emacs, or from other front ends is possible. First start GDB.

Tell it to open the UMLPOOL/swan/linux program.

Note the PID of GDB:

```
marajade-[projects/freeswan/mgmt/planning] mcr 1029 %ps ax | grep gdb
1659 pts/9      SN      0:00 /usr/bin/gdb -fullname -cd /mara4/freeswan/kernpatch/UMLPOOL/swan/ lin
```

Set the following in the environment:

```
UML_east_OPT="debug gdb-pid=1659"
```

Then start the user-mode-linux in the test scheme you wish:

```
marajade-[kernpatch/testing/klips/east-icmp-02] mcr 1220 %../utils/runme.sh
```

The user-mode-linux will stop on boot, giving you a chance to attach to the process:

```
(gdb) file linux
Reading symbols from linux...done.
(gdb) attach 1
Attaching to program: /mara4/freeswan/kernpatch/UMLPOOL/swan/linux, process 1
0xa0118bc1 in kill () at hostfs_kern.c:770
```

At this point, break points should be created as appropriate.

## Other notes about debugging

If you are running a standard test, after all the packets are sent, the UML will be shutdown. This can cause problems, because the UML may get terminated while you are debugging.

The environment variable NETJIGWAITUSER can be set to "waituser". If so, then the testing system will prompt before exiting the test.

The environment variable UML\_GETTY if set, will cause each UML to spawn a getty on /dev/tty1, and will wait for it to exit before continuing.

# User-Mode-Linux mysteries

- running more than one UML of the same name (e.g. "west") can cause problems.
- running more than one UML from the same root file system is not a good idea.
- all this means that running "make check" twice on the same machine is probably not a good idea.
- occasionally, UMLs will get stuck. This can happen like: 15134 ? T 0:00  
/spare/hugh/uml/uml2.4.18-sept5/umlbuild/east/linux (east) [/bin/sh] 15138 ? T 0:00  
/spare/hugh/uml/uml2.4.18-sept5/umlbuild/east/linux (east) [halt] these will need to be killed. Note that they are in "T" racing mode.
- UMLs can also hang, and will report "Tracing myself and I can't get out". This is a bug in UML. There are ways to find out what is going on and report this to the UML people, but we don't know the magic right now.

## Getting more info from `uml_netjig`

`uml_netjig` can be compiled with a built-in `tcpdump`. This uses not-yet-released code from [www.tcpdump.org](http://www.tcpdump.org). Please see the instructions in `testing/utls/uml_netjig/Makefile`.

# History and politics of cryptography

Cryptography has a long and interesting history, and has been the subject of considerable political controversy.

## Introduction

### History

The classic book on the history of cryptography is David Kahn's The Codebreakers. It traces codes and codebreaking from ancient Egypt to the 20th century.

Diffie and Landau Privacy on the Line: The Politics of Wiretapping and Encryption covers the history from the First World War to the 1990s, with an emphasis on the US.

### World War II

During the Second World War, the British "Ultra" project achieved one of the greatest intelligence triumphs in the history of warfare, breaking many Axis codes. One major target was the Enigma cipher machine, a German device whose users were convinced it was unbreakable. The American "Magic" project had some similar triumphs against Japanese codes.

There are many books on this period. See our bibliography for several. Two I particularly like are:

- Andrew Hodges has done a superb biography of Alan Turing, a key player among the Ultra codebreakers. Turing was also an important computer pioneer. The terms Turing test and Turing machine are named for him, as is the ACM's highest technical award.
- Neal Stephenson's Cryptonomicon is a novel with cryptography central to the plot. Parts of it take place during WW II, other parts today.

Bletchley Park, where much of the Ultra work was done, now has a museum and a web site.

The Ultra work introduced three major innovations.

- The first break of Enigma was achieved by Polish Intelligence in 1931. Until then most code-breakers had been linguists, but a different approach was needed to break machine ciphers. Polish Intelligence recruited bright young mathematicians to crack the "unbreakable" Enigma. When war came in 1939, the Poles told their allies about this, putting Britain on the road to Ultra. The British also adopted a mathematical approach.
- Machines were extensively used in the attacks. First the Polish "Bombe" for attacking Enigma, then British versions of it, then machines such as Collosus for attacking other codes. By the end of the war, some of these machines were beginning to closely resemble digital computers. After the war, a team at Manchester University, several old Ultra hands included, built one of the world's first actual general-purpose digital computers.
- Ultra made codebreaking a large-scale enterprise, producing intelligence on an industrial scale. This was not a "black chamber", not a hidden room in some obscure government building with a small crew of code-breakers. The whole operation — from wholesale interception of enemy communications by stations around the world, through large-scale code-breaking and analysis of the decrypted material (with an enormous set of files for cross-referencing), to delivery of intelligence to

field commanders — was huge, and very carefully managed.

So by the end of the war, Allied code-breakers were expert at large-scale mechanised code-breaking. The payoffs were enormous.

### Postwar and Cold War

The wartime innovations were enthusiastically adopted by post-war and Cold War signals intelligence agencies. Presumably many nations now have some agency capable of sophisticated attacks on communications security, and quite a few engage in such activity on a large scale.

America's NSA, for example, is said to be both the world's largest employer of mathematicians and the world's largest purchaser of computer equipment. Such claims may be somewhat exaggerated, but beyond doubt the NSA — and similar agencies in other countries — have some excellent mathematicians, lots of powerful computers, sophisticated software, and the organisation and funding to apply them on a large scale. Details of the NSA budget are secret, but there are some published estimates.

Changes in the world's communications systems since WW II have provided these agencies with new targets. Cracking the codes used on an enemy's military or diplomatic communications has been common practice for centuries. Extensive use of radio in war made large-scale attacks such as Ultra possible. Modern communications make it possible to go far beyond that. Consider listening in on cell phones, or intercepting electronic mail, or tapping into the huge volumes of data on new media such as fiber optics or satellite links. None of these targets existed in 1950. All of them can be attacked today, and almost certainly are being attacked.

The Ultra story was not made public until the 1970s. Much of the recent history of codes and code-breaking has not been made public, and some of it may never be. Two important books are:

- Bamford's The Puzzle Palace, a history of the NSA
- Hager's Secret Power, about the Echelon system — the US, UK, Canada, Australia and New Zealand co-operating to monitor much of the world's communications.

Note that these books cover only part of what is actually going on, and then only the activities of nations open and democratic enough that (some of) what they are doing can be discovered. A full picture, including:

- actions of the English-speaking democracies not covered in those books
- actions of other more-or-less sane governments
- the activities of various more-or-less insane governments
- possibilities for unauthorized action by government employees
- possible actions by large non-government organisations: corporations, criminals, or conspiracies

might be really frightening.

### Recent history -- the crypto wars

Until quite recently, cryptography was primarily a concern of governments, especially of the military, of spies, and of diplomats. Much of it was extremely secret.

In recent years, that has changed a great deal. With computers and networking becoming ubiquitous, cryptography is now important to almost everyone. Among the developments since the 1970s:

## Introduction to FreeS/WAN

- The US gov't established the Data Encryption Standard, DES, a block cipher for cryptographic protection of unclassified documents.
- DES also became widely used in industry, especially regulated industries such as banking.
- Other nations produced their own standards, such as GOST in the Soviet Union.
- Public key cryptography was invented by Diffie and Hellman.
- Academic conferences such as Crypto and Eurocrypt began.
- Several companies began offering cryptographic products: RSA, PGP, the many vendors with PKI products, ...
- Cryptography appeared in other products: operating systems, word processors, ...
- Network protocols based on crypto were developed: SSH, SSL, IPsec, ...
- Cryptography came into widespread use to secure bank cards, terminals, ...
- The US government replaced DES with the much stronger Advanced Encryption Standard, AES

This has led to a complex ongoing battle between various mainly government groups wanting to control the spread of crypto and various others, notably the computer industry and the cypherpunk crypto advocates, wanting to encourage widespread use.

Steven Levy has written a fine history of much of this, called Crypto: How the Code rebels Beat the Government — Saving Privacy in the Digital Age.

The FreeS/WAN project is to a large extent an outgrowth of cypherpunk ideas. Our reasons for doing the project can be seen in these quotes from the Cypherpunk Manifesto:

Privacy is necessary for an open society in the electronic age. ...

We cannot expect governments, corporations, or other large, faceless organizations to grant us privacy out of their beneficence. It is to their advantage to speak of us, and we should expect that they will speak. ...

We must defend our own privacy if we expect to have any. ...

Cypherpunks write code. We know that someone has to write software to defend privacy, and since we can't get privacy unless we all do, we're going to write it. We publish our code so that our fellow Cypherpunks may practice and play with it. Our code is free for all to use, worldwide. We don't much care if you don't approve of the software we write. We know that software can't be destroyed and that a widely dispersed system can't be shut down.

Cypherpunks deplore regulations on cryptography, for encryption is fundamentally a private act. ...

For privacy to be widespread it must be part of a social contract. People must come and together deploy these systems for the common good. ...

To quote project leader John Gilmore:

We are literally in a race between our ability to build and deploy technology, and their ability to build and deploy laws and treaties. Neither side is likely to back down or wise up until it has definitively lost the race.

If FreeS/WAN reaches its goal of making opportunistic encryption widespread so that secure communication can become the default for a large part of the net, we will have struck a major blow.

## Politics

The political problem is that nearly all governments want to monitor their enemies' communications, and some want to monitor their citizens. They may be very interested in protecting some of their own communications, and often some types of business communication, but not in having everyone able to communicate securely. They therefore attempt to restrict availability of strong cryptography as much as possible.

Things various governments have tried or are trying include:

- Echelon, a monitor-the-world project of the US, UK, NZ, Australian and Canadian signals intelligence agencies. See this collection of links and this story on the French Parliament's reaction.
- Others governments may well have their own Echelon-like projects. To quote the Dutch Minister of Defense, as reported in a German magazine:

The government believes not only the governments associated with Echelon are able to intercept communication systems, but that it is an activity of the investigative authorities and intelligence services of many countries with governments of different political signature.

Even if they have nothing on the scale of Echelon, most intelligence agencies and police forces certainly have some interception capability.

- NSA tapping of submarine communication cables, described in this article
- A proposal for international co-operation on Internet surveillance.
- Alleged sabotage of security products by the NSA (the US signals intelligence agency).
- The German armed forces and some government departments will stop using American software for fear of NSA "back doors", according to this news story.
- The British Regulation of Investigatory Powers bill. See this web page, and perhaps this cartoon.
- A Russian ban on cryptography
- Chinese controls on net use.
- The FBI's carnivore system for covert searches of email. See this news coverage and this risk assessment. The government had an external review of some aspects of this system done. See this analysis of that review. Possible defenses against Carnivore include:
  - ♦ PGP for end-to-end mail encryption
  - ♦ secure sendmail for server-to-server encryption
  - ♦ IPsec encryption on the underlying IP network
- export laws restricting strong cryptography as a munition. See discussion below.
- various attempts to convince people that fundamentally flawed cryptography, such as encryption with a back door for government access to data or with inadequate key lengths, was adequate for their needs.

Of course governments are by no means the only threat to privacy and security on the net. Other threats include:

- industrial espionage, as for example in this news story
- attacks by organised criminals, as in this large-scale attack
- collection of personal data by various companies.
  - ♦ for example, consider the various corporate winners of Privacy International's Big Brother Awards.
  - ♦ Zero Knowledge sell tools to defend against this

## Introduction to FreeS/WAN

- individuals may also be a threat in a variety of ways and for a variety of reasons
- in particular, an individual with access to government or industry data collections could do considerable damage using that data in unauthorized ways.

One study enumerates threats and possible responses for small and medium businesses. VPNs are a key part of the suggested strategy.

We consider privacy a human right. See the UN's Universal Declaration of Human Rights, article twelve:

No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honor and reputation. Everyone has the right to the protection of the law against such interference or attacks.

Our objective is to help make privacy possible on the Internet using cryptography strong enough not even those well-funded government agencies are likely to break it. If we can do that, the chances of anyone else breaking it are negliible.

## Links

Many groups are working in different ways to defend privacy on the net and elsewhere. Please consider contributing to one or more of these groups:

- the EFF's Privacy Now! campaign
- the Global Internet Liberty Campaign
- Computer Professionals for Social Responsibility

For more on these issues see:

- Steven Levy (Newsweek's chief technology writer and author of the classic "Hackers") new book Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age
- Simson Garfinkel (Boston Globe columnist and author of books on PGP and Unix Security) book Database Nation: the death of privacy in the 21st century

There are several collections of crypto quotes on the net.

See also the bibliography and our list of web references on cryptography law and policy.

## Outline of this section

The remainder of this section includes two pieces of writing by our project leader

- his rationale for starting this
- another discussion of project goals

and discussions of:

- why we do not use DES
- cryptography export laws
- why government access to keys is not a good idea
- the myth that short keys are adequate for some security requirements

and a section on [press coverage of FreeS/WAN](#).

## From our project leader

FreeS/WAN project founder John Gilmore wrote a web page about why we are doing this. The version below is slightly edited, to fit this format and to update some links. For a version without these edits, see his [home page](#).

### Swan: Securing the Internet against Wiretapping

My project for 1996 was to **secure 5% of the Internet traffic against passive wiretapping**. It didn't happen in 1996, so I'm still working on it in 1997, 1998, and 1999! If we get 5% in 1999 or 2000, we can secure 20% the next year, against both active and passive attacks; and 80% the following year. Soon the whole Internet will be private and secure. The project is called S/WAN or S/Wan or Swan for Secure Wide Area Network; since it's free software, we call it FreeSwan to distinguish it from various commercial implementations. [RSA](#) came up with the term "S/WAN". Our main web site is at <http://www.freeswan.org/>. Want to help?

The idea is to deploy PC-based boxes that will sit between your local area network and the Internet (near your firewall or router) which opportunistically encrypt your Internet packets. Whenever you talk to a machine (like a Web site) that doesn't support encryption, your traffic goes out "in the clear" as usual. Whenever you connect to a machine that does support this kind of encryption, this box automatically encrypts all your packets, and decrypts the ones that come in. In effect, each packet gets put into an "envelope" on one side of the net, and removed from the envelope when it reaches its destination. This works for all kinds of Internet traffic, including Web access, Telnet, FTP, email, IRC, Usenet, etc.

The encryption boxes are standard PC's that use freely available Linux software that you can download over the Internet or install from a cheap CDROM.

This wasn't just my idea; lots of people have been working on it for years. The encryption protocols for these boxes are called [IPSEC \(IP Security\)](#). They have been developed by the [IP Security Working Group](#) of the [Internet Engineering Task Force](#), and will be a standard part of the next major version of the Internet protocols ([IPv6](#)). For today's (IP version 4) Internet, they are an option.

The [Internet Architecture Board](#) and [Internet Engineering Steering Group](#) have taken a [strong stand](#) that the Internet should use powerful encryption to provide security and privacy. I think these protocols are the best chance to do that, because they can be deployed very easily, without changing your hardware or software or retraining your users. They offer the best security we know how to build, using the Triple-DES, RSA, and Diffie-Hellman algorithms.

This "opportunistic encryption box" offers the "fax effect". As each person installs one for their own use, it becomes more valuable for their neighbors to install one too, because there's one more person to use it with. The software automatically notices each newly installed box, and doesn't require a network administrator to reconfigure it. Instead of "virtual private networks" we have a "REAL private network"; we add privacy to the real network instead of layering a manually-maintained virtual network on top of an insecure Internet.

### Deployment of IPSEC

The US government would like to control the deployment of IP Security with its [crypto export laws](#). This isn't a problem for my effort, because the cryptographic work is happening outside the United States. A foreign philanthropist, and others, have donated the resources required to add these protocols to the Linux operating

## Introduction to FreeS/WAN

system. Linux is a complete, freely available operating system for IBM PC's and several kinds of workstation, which is compatible with Unix. It was written by Linus Torvalds, and is still maintained by a talented team of expert programmers working all over the world and coordinating over the Internet. Linux is distributed under the GNU Public License, which gives everyone the right to copy it, improve it, give it to their friends, sell it commercially, or do just about anything else with it, without paying anyone for the privilege.

Organizations that want to secure their network will be able to put two Ethernet cards into an IBM PC, install Linux on it from a \$30 CDROM or by downloading it over the net, and plug it in between their Ethernet and their Internet link or firewall. That's all they'll have to do to encrypt their Internet traffic everywhere outside their own local area network.

Travelers will be able to run Linux on their laptops, to secure their connection back to their home network (and to everywhere else that they connect to, such as customer sites). Anyone who runs Linux on a standalone PC will also be able to secure their network connections, without changing their application software or how they operate their computer from day to day.

There will also be numerous commercially available firewalls that use this technology. RSA Data Security is coordinating the S/Wan (Secure Wide Area Network) project among more than a dozen vendors who use these protocols. There's a compatability chart that shows which vendors have tested their boxes against which other vendors to guarantee interoperability.

Eventually it will also move into the operating systems and networking protocol stacks of major vendors. This will probably take longer, because those vendors will have to figure out what they want to do about the export controls.

### Current status

My initial goal of securing 5% of the net by Christmas '96 was not met. It was an ambitious goal, and inspired me and others to work hard, but was ultimately too ambitious. The protocols were in an early stage of development, and needed a lot more protocol design before they could be implemented. As of April 1999, we have released version 1.0 of the software (freeswan-1.0.tar.gz), which is suitable for setting up Virtual Private Networks using shared secrets for authentication. It does not yet do opportunistic encryption, or use DNSSEC for authentication; those features are coming in a future release.

#### *Protocols*

The low-level encrypted packet formats are defined. The system for publishing keys and providing secure domain name service is defined. The IP Security working group has settled on an NSA-sponsored protocol for key agreement (called ISAKMP/Oakley), but it is still being worked on, as the protocol and its documentation is too complex and incomplete. There are prototype implementations of ISAKMP. The protocol is not yet defined to enable opportunistic encryption or the use of DNSSEC keys.

#### *Linux Implementation*

The Linux implementation has reached its first major release and is ready for production use in manually-configured networks, using Linux kernel version 2.0.36.

#### *Domain Name System Security*

There is now a release of BIND 8.2 that includes most DNS Security features.

The first prototype implementation of Domain Name System Security was funded by DARPA as part of their Information Survivability program. Trusted Information Systems wrote a modified version of BIND, the widely-used Berkeley implementation of the Domain Name System.

TIS, ISC, and I merged the prototype into the standard version of BIND. The first production version that supports KEY and SIG records is **bind-4.9.5**. This or any later version of BIND will do for publishing keys. It is available from the [Internet Software Consortium](#). This version of BIND is not export-controlled since it does not contain any cryptography. Later releases starting with BIND 8.2 include cryptography for authenticating DNS records, which is also exportable. Better documentation is needed.

### Why?

Because I can. I have made enough money from several successful startup companies, that for a while I don't have to work to support myself. I spend my energies and money creating the kind of world that I'd like to live in and that I'd like my (future) kids to live in. Keeping and improving on the civil rights we have in the United States, as we move more of our lives into cyberspace, is a particular goal of mine.

### What You Can Do

*Install the latest BIND at your site.*

You won't be able to publish any keys for your domain, until you have upgraded your copy of BIND. The thing you really need from it is the new version of *named*, the Name Daemon, which knows about the new KEY and SIG record types. So, download it from the [Internet Software Consortium](#) and install it on your name server machine (or get your system administrator, or Internet Service Provider, to install it). Both your primary DNS site and all of your secondary DNS sites will need the new release before you will be able to publish your keys. You can tell which sites this is by running the Unix command "dig MYDOMAIN ns" and seeing which sites are mentioned in your NS (name server) records.

*Set up a Linux system and run a 2.0.x kernel on it*

Get a machine running Linux (say the 5.2 release from [Red Hat](#)). Give the machine two Ethernet cards.

*Install the Linux IPSEC (Freeswan) software*

If you're an experienced sysadmin or Linux hacker, install the freeswan-1.0 release, or any later release or snapshot. These releases do NOT provide automated "opportunistic" operation; they must be manually configured for each site you wish to encrypt with.

*Get on the linux-ipsec mailing list*

The discussion forum for people working on the project, and testing the code and documentation, is: [linux-ipsec@clinet.fi](mailto:linux-ipsec@clinet.fi). To join this mailing list, send email to [linux-ipsec-REQUEST@clinet.fi](mailto:linux-ipsec-REQUEST@clinet.fi) containing a line of text that says "subscribe linux-ipsec". (You can later get off the mailing list the same way — just send "unsubscribe linux-ipsec").

*Check back at this web page every once in a while*

I update this page periodically, and there may be new information in it that you haven't seen. My intent is to send email to the mailing list when I update the page in any significant way, so subscribing to the list is an alternative.

Would you like to help? I can use people who are willing to write documentation, install early releases for testing, write cryptographic code outside the United States, sell pre-packaged software or systems including this technology, and teach classes for network administrators who want to install this technology. To offer to help, send me email at [gnu@toad.com](mailto:gnu@toad.com). Tell me what country you live in and what your citizenship is (it matters due to the export control laws; personally I don't care). Include a copy of your resume and the URL of your home page. Describe what you'd like to do for the project, and what you're uniquely qualified for. Mention what other volunteer projects you've been involved in (and how they worked out). Helping out will require that you be able to commit to doing particular things, meet your commitments, and be responsive by

email. Volunteer projects just don't work without those things.

### Related projects

#### *IPSEC for NetBSD*

This prototype implementation of the IP Security protocols is for another free operating system.

[Download BSDipsec.tar.gz.](#)

#### *IPSEC for OpenBSD*

This prototype implementation of the IP Security protocols is for yet another free operating system. It is directly integrated into the OS release, since the OS is maintained in Canada, which has freedom of speech in software.

## Stopping wholesale monitoring

From a message project leader John Gilmore posted to the mailing list:

John Denker wrote:

```
> Indeed there are several ways in which the documentation overstates the
> scope of what this project does -- starting with the name
> FreeS/WAN. There's a big difference between having an encrypted IP tunnel
> versus having a Secure Wide-Area Network. This software does a fine job of
> the former, which is necessary but not sufficient for the latter.
```

The goal of the project is to make it very hard to tap your wide area communications. The current system provides very good protection against passive attacks (wiretapping and those big antenna farms). Active attacks, which involve the intruder sending packets to your system (like packets that break into sendmail and give them a root shell :-)) are much harder to guard against. Active attacks that involve sending people (breaking into your house and replacing parts of your computer with ones that transmit what you're doing) are also much harder to guard against. Though we are putting effort into protecting against active attacks, it's a much bigger job than merely providing strong encryption. It involves general computer security, and general physical security, which are two very expensive problems for even a site to solve, let alone to build into a whole society.

The societal benefit of building an infrastructure that protects well against passive attacks is that it makes it much harder to do undetected bulk monitoring of the population. It's a defense against police-states, not against policemen.

Policemen can put in the effort required to actively attack sites that they have strong suspicions about. But police states won't be able to build systems that automatically monitor everyone's communications. Either they will be able to monitor only a small subset of the populace (by targeting those who screwed up their passive security), or their monitoring activities will be detectable by those monitored (active attacks leave packet traces or footprints), which can then be addressed through the press and through political means if they become too widespread.

FreeS/WAN does not protect very well against traffic analysis, which is a kind of widespread police-state style monitoring that still reveals significant information (who's talking to who) without revealing the contents of what was said. Defenses against traffic analysis are an open research problem. Zero Knowledge Systems is

## Introduction to FreeS/WAN

actively deploying a system designed to thwart it, designed by Ian Goldberg. The jury is out on whether it actually works; a lot more experience with it will be needed.

Notes on things mentioned in that message:

- Denker is a co-author of a paper on a large FreeS/WAN application.
- Information on Zero Knowledge is on their web site. Their Freedom product, designed to provide untracable pseudonyms for use on the net, is no longer marketed.
- Another section of our documentation discusses ways to resist traffic analysis.

## Government promotion of weak crypto

Various groups, especially governments and especially the US government, have a long history of advocating various forms of bogus security.

We regard bogus security as extremely dangerous. If users are deceived into relying on bogus security, then they may be exposed to large risks. They would be better off having no security and knowing it. At least then they would be careful about what they said.

*Avoiding bogus security is a key design criterion for everything we do in FreeS/WAN.* The most conspicuous example is our refusal to support single DES. Other IPsec "features" which we do not implement are discussed in our compatibility document.

## Escrowed encryption

Various governments have made persistent attempts to encourage or mandate "escrowed encryption", also called "key recovery", or GAK for "government access to keys". The idea is that cryptographic keys be held by some third party and turned over to law enforcement or security agencies under some conditions.

Mary had a little key - she kept it in escrow,  
and every thing that Mary said,  
the feds were sure to know.

A crypto quotes page attributes this to Sam Simpson.

There is an excellent paper available on Risks of Escrowed Encryption, from a group of cryptographic luminaries which included our project leader.

Like any unnecessary complication, GAK tends to weaken security of any design it infects. For example:

- Matt Blaze found a fatal flaw in the US government's Clipper chip shortly after design information became public. See his paper "Protocol Failure in the Escrowed Encryption Standard" on his papers page.
- a rather nasty bug was found in the "additional decryption keys" "feature" of some releases of PGP

FreeS/WAN does not support escrowed encryption, and never will.

## Limited key lengths

Various governments, and some vendors, have also made persistent attempts to convince people that:

- weak systems are sufficient for some data
- strong cryptography should be reserved for cases where the extra overheads are justified

*This is utter nonsense.*

Weak systems touted include:

- the ludicrously weak (deliberately crippled) 40-bit ciphers that until recently were all various export laws allowed
- 56-bit single DES, discussed below
- 64-bit symmetric ciphers and 512-bit RSA, the maximums for unrestricted export under various current laws

The notion that choice of ciphers or keysize should be determined by a trade-off between security requirements and overheads is pure bafflegab.

- For most symmetric ciphers, it is simply a lie. Any block cipher has some natural maximum keysize inherent in the design — 128 bits for IDEA or CAST-128, 256 for Serpent or Twofish, 448 for Blowfish and 2048 for RC4. Using a key size smaller than that limit gives *exactly zero* savings in overhead. The crippled 40-bit or 64-bit version of the cipher provides *no advantage whatsoever*.
- AES uses 10 rounds with 128-bit keys, 12 rounds for 192-bit and 14 rounds for 256-bit, so there actually is a small difference in overhead, but not enough to matter in most applications.
- For triple DES there is a grain of truth in the argument. 3DES is indeed three times slower than single DES. However, the solution is not to use the insecure single DES, but to pick a faster secure cipher. CAST-128, Blowfish and the AES candidate ciphers are all considerably faster in software than DES (let alone 3DES!), and apparently secure.
- For public key techniques, there are extra overheads for larger keys, but they generally do not affect overall performance significantly. Practical public key applications are usually hybrid systems in which the bulk of the work is done by a symmetric cipher. The effect of increasing the cost of the public key operations is typically negligible because the public key operations use only a tiny fraction of total resources.

For example, suppose public key operations use 1% of the time in a hybrid system and you triple the cost of public key operations. The cost of symmetric cipher operations is unchanged at 99% of the original total cost, so the overall effect is a jump from  $99 + 1 = 100$  to  $99 + 3 = 102$ , a 2% rise in system cost.

In short, *there has never been any technical reason to use inadequate ciphers*. The only reason there has ever been for anyone to use such ciphers is that government agencies want weak ciphers used so that they can crack them. The alleged savings are simply propaganda.

Mary had a little key (It's all she could export),  
and all the email that she sent was opened at the Fort.

A crypto quotes page attributes this to Ron Rivest. NSA headquarters is at Fort Meade, Maryland.

## Introduction to FreeS/WAN

Our policy in FreeS/WAN is to use only cryptographic components with adequate keylength and no known weaknesses.

- We do not implement single DES because it is clearly insecure, so implementing it would violate our policy of avoiding bogus security. Our default cipher is 3DES
- Similarly, we do not implement the 768-bit Group 1 for Diffie-Hellman key negotiation. We provide only the 1024-bit Group 2 and 1536-bit Group 5.

Detailed discussion of which IPsec features we implement or omit is in our compatibility document.

These decisions imply that we cannot fully conform to the IPsec RFCs, since those have DES as the only required cipher and Group 1 as the only required DH group. (In our view, the standards were subverted into offering bogus security.) Fortunately, we can still interoperate with most other IPsec implementations since nearly all implementers provide at least 3DES and Group 2 as well.

We hope that eventually the RFCs will catch up with our (and others') current practice and reject dubious components. Some of our team and a number of others are working on this in IETF working groups.

### Some real trade-offs

Of course, making systems secure does involve costs, and trade-offs can be made between cost and security. However, the real trade-offs have nothing to do with using weaker ciphers.

There can be substantial hardware and software costs. There are often substantial training costs, both to train administrators and to increase user awareness of security issues and procedures. There are almost always substantial staff or contracting costs.

Security takes staff time for planning, implementation, testing and auditing. Some of the issues are subtle; you need good (hence often expensive) people for this. You also need people to monitor your systems and respond to problems. The best safe ever built is insecure if an attacker can work on it for days without anyone noticing. Any computer is insecure if the administrator is "too busy" to check the logs.

Moreover, someone in your organisation (or on contract to it) needs to spend considerable time keeping up with new developments. EvilDoers *will* know about new attacks shortly after they are found. You need to know about them before your systems are attacked. If your vendor provides a patch, you need to apply it. If the vendor does nothing, you need to complain or start looking for another vendor.

For a fairly awful example, see this report. In that case over a million credit card numbers were taken from e-commerce sites, using security flaws in Windows NT servers. Microsoft had long since released patches for most or all of the flaws, but the site administrators had not applied them.

At an absolute minimum, you must do something about such issues *before* an exploitation tool is posted to the net for downloading by dozens of "script kiddies". Such a tool might appear at any time from the announcement of the security hole to several months later. Once it appears, anyone with a browser and an attitude can break any system whose administrators have done nothing about the flaw.

Compared to those costs, cipher overheads are an insignificant factor in the cost of security.

The only thing using a weak cipher can do for you is to cause all your other investment to be wasted.

## Cryptography Export Laws

Many nations restrict the export of cryptography and some restrict its use by their citizens or others within their borders.

### US Law

US laws, as currently interpreted by the US government, forbid export of most cryptographic software from the US in machine-readable form without government permission. In general, the restrictions apply even if the software is widely-disseminated or public-domain and even if it came from outside the US originally. Cryptography is legally a munition and export is tightly controlled under the EAR Export Administration Regulations.

If you are a US citizen, your brain is considered US territory no matter where it is physically located at the moment. The US believes that its laws apply to its citizens everywhere, not just within the US. Providing technical assistance or advice to foreign "munitions" projects is illegal. The US government has very little sense of humor about this issue and does not consider good intentions to be sufficient excuse. Beware.

The official website for these regulations is run by the Commerce Department's Bureau of Export Administration (BXA).

The Bernstein case challenges the export restrictions on Constitutional grounds. Code is speech so restrictions on export of code violate the First Amendment's free speech provisions. This argument has succeeded in two levels of court so far. It is quite likely to go on to the Supreme Court.

The regulations were changed substantially in January 2000, apparently as a government attempt to get off the hook in the Bernstein case. It is now legal to export public domain source code for encryption, provided you notify the BXA.

There are, however, still restrictions in force. Moreover, the regulations can still be changed again whenever the government chooses to do so. Short of a Supreme Court ruling (in the Bernstein case or another) that overturns the regulations completely, the problem of export regulation is not likely to go away in the foreseeable future.

### US contributions to FreeS/WAN

The FreeS/WAN project ***cannot accept software contributions, not even small bug fixes, from US citizens or residents***. We want it to be absolutely clear that our distribution is not subject to US export law. Any contribution from an American might open that question to a debate we'd prefer to avoid. It might also put the contributor at serious legal risk.

Of course Americans can still make valuable contributions (many already have) by reporting bugs, or otherwise contributing to discussions, on the project mailing list. Since the list is public, this is clearly constitutionally protected free speech.

Note, however, that the export laws restrict Americans from providing technical assistance to foreign "munitions" projects. The government might claim that private discussions or correspondence with FreeS/WAN developers were covered by this. It is not clear what the courts would do with such a claim, so we strongly encourage Americans to use the list rather than risk the complications.

## What's wrong with restrictions on cryptography

Some quotes from prominent cryptography experts:

The real aim of current policy is to ensure the continued effectiveness of US information warfare assets against individuals, businesses and governments in Europe and elsewhere.  
Ross Anderson, Cambridge University

If the government were honest about its motives, then the debate about crypto export policy would have ended years ago.  
Bruce Schneier, Counterpane Systems

The NSA regularly lies to people who ask it for advice on export control. They have no reason not to; accomplishing their goal by any legal means is fine by them. Lying by government employees is legal.  
John Gilmore.

The Internet Architecture Board (IAB) and the Internet Engineering Steering Group (IESG) made a strong statement in favour of worldwide access to strong cryptography. Essentially the same statement is in the appropriately numbered RFC 1984. Two critical paragraphs are:

... various governments have actual or proposed policies on access to cryptographic technology ...

- (a) ... export controls ...
- (b) ... short cryptographic keys ...
- (c) ... keys should be in the hands of the government or ...
- (d) prohibit the use of cryptology ...

We believe that such policies are against the interests of consumers and the business community, are largely irrelevant to issues of military security, and provide only a marginal or illusory benefit to law enforcement agencies, ...

The IAB and IESG would like to encourage policies that allow ready access to uniform strong cryptographic technology for all Internet users in all countries.

Our goal in the FreeS/WAN project is to build just such "strong cryptographic technology" and to distribute it "for all Internet users in all countries".

More recently, the same two bodies (IESG and IAB) have issued RFC 2804 on why the IETF should not build wiretapping capabilities into protocols for the convenience of security or law enforcement agencies. The abstract from that document is:

The Internet Engineering Task Force (IETF) has been asked to take a position on the inclusion into IETF standards-track documents of functionality designed to facilitate wiretapping.

This memo explains what the IETF thinks the question means, why its answer is "no", and what that answer means.

## Introduction to FreeS/WAN

A quote from the debate leading up to that RFC:

We should not be building surveillance technology into standards. Law enforcement was not supposed to be easy. Where it is easy, it's called a police state.  
Jeff Schiller of MIT, in a discussion of FBI demands for wiretap capability on the net, as quoted by Wired.

The Raven mailing list was set up for this IETF discussion.

Our goal is to go beyond that RFC and prevent Internet wiretapping entirely.

## The Wassenaar Arrangement

Restrictions on the export of cryptography are not just US policy, though some consider the US at least partly to blame for the policies of other nations in this area.

A number of countries:

Argentina, Australia, Austria, Belgium, Bulgaria, Canada, Czech Republic, Denmark, Finland, France, Germany, Greece, Hungary, Ireland, Italy, Japan, Luxembourg, Netherlands, New Zealand, Norway, Poland, Portugal, Republic of Korea, Romania, Russian Federation, Slovak Republic, Spain, Sweden, Switzerland, Turkey, Ukraine, United Kingdom and United States

have signed the Wassenaar Arrangement which restricts export of munitions and other tools of war. Cryptographic software is covered there.

Wassenaar details are available from the Wassenaar Secretariat, and elsewhere in a more readable HTML version.

For a critique see the GILC site:

The Global Internet Liberty Campaign (GILC) has begun a campaign calling for the removal of cryptography controls from the Wassenaar Arrangement.

The aim of the Wassenaar Arrangement is to prevent the build up of military capabilities that threaten regional and international security and stability . . .

There is no sound basis within the Wassenaar Arrangement for the continuation of any export controls on cryptographic products.

We agree entirely.

An interesting analysis of Wassenaar can be found on the cyber-rights.org site.

## Export status of Linux FreeS/WAN

We believe our software is entirely exempt from these controls since the Wassenaar General Software Note says:

The Lists do not control "software" which is either:

## Introduction to FreeS/WAN

1. Generally available to the public by . . . retail . . . or
2. "In the public domain".

There is a note restricting some of this, but it is a sub-heading under point 1, so it appears not to apply to public domain software.

Their glossary defines "In the public domain" as:

. . . "technology" or "software" which has been made available without restrictions upon its further dissemination.

N.B. Copyright restrictions do not remove "technology" or "software" from being "in the public domain".

We therefore believe that software freely distributed under the GNU Public License, such as Linux FreeS/WAN, is exempt from Wassenaar restrictions.

Most of the development work is being done in Canada. Our understanding is that the Canadian government accepts this interpretation.

- A web statement of Canadian policy is available from the Department of Foreign Affairs and International Trade.
- Another document from that department states that public domain software is exempt from the export controls.
- A researcher's analysis of Canadian policy is also available.

Recent copies of the freely modifiable and distributable source code exist in many countries. Citizens all over the world participate in its use and evolution, and guard its ongoing distribution. Even if Canadian policy were to change, the software would continue to evolve in countries which do not restrict exports, and would continue to be imported from there into unfree countries. "The Net culture treats censorship as damage, and routes around it."

## Help spread IPsec around

You can help. If you don't know of a Linux FreeS/WAN archive in your own country, please download it now to your personal machine, and consider making it publicly accessible if that doesn't violate your own laws. If you have the resources, consider going one step further and setting up a mirror site for the whole munitions Linux crypto software archive.

If you make Linux CD-ROMs, please consider including this code, in a way that violates no laws (in a free country, or in a domestic-only CD product).

Please send a note about any new archive mirror sites or CD distributions to [linux-ipsec@clinet.fi](mailto:linux-ipsec@clinet.fi) so we can update the documentation.

Lists of current mirror sites and of distributions which include FreeS/WAN are in our introduction section.

## DES is Not Secure

DES, the *Data Encryption Standard*, can no longer be considered secure. While no major flaws in its innards are known, it is fundamentally inadequate because its **56-bit key is too short**. It is vulnerable to brute-force search of the whole key space, either by large collections of general-purpose machines or even more quickly by specialized hardware. Of course this also applies to **any other cipher with only a 56-bit key**. The only reason anyone could have for using a 56 or 64-bit key is to comply with various export laws intended to ensure the use of breakable ciphers.

Non-government cryptologists have been saying DES's 56-bit key was too short for some time — some of them were saying it in the 70's when DES became a standard — but the US government has consistently ridiculed such suggestions.

A group of well-known cryptographers looked at key lengths in a 1996 paper. They suggested a *minimum* of 75 bits to consider an existing cipher secure and a *minimum of 90 bits for new ciphers*. More recent papers, covering both symmetric and public key systems are at cryptosavvy.com and rsa.com. For all algorithms, the minimum keylengths recommended in such papers are significantly longer than the maximums allowed by various export laws.

In a 1998 ruling, a German court described DES as "out-of-date and not safe enough" and held a bank liable for using it.

## Dedicated hardware breaks DES in a few days

The question of DES security has now been settled once and for all. In early 1998, the Electronic Frontier Foundation built a DES-cracking machine. It can find a DES key in an average of a few days' search. The details of all this, including complete code listings and complete plans for the machine, have been published in Cracking DES, by the Electronic Frontier Foundation.

That machine cost just over \$200,000 to design and build. "Moore's Law" is that machines get faster (or cheaper, for the same speed) by roughly a factor of two every 18 months. At that rate, their \$200,000 in 1998 becomes \$50,000 in 2001.

However, Moore's Law is not exact and the \$50,000 estimate does not allow for the fact that a copy based on the published EFF design would cost far less than the original. We cannot say exactly what such a cracker would cost today, but it would likely be somewhere between \$10,000 and \$100,000.

A large corporation could build one of these out of petty cash. The cost is low enough for a senior manager to hide it in a departmental budget and avoid having to announce or justify the project. Any government agency, from a major municipal police force up, could afford one. Or any other group with a respectable budget — criminal organisations, political groups, labour unions, religious groups, ... Or any millionaire with an obsession or a grudge, or just strange taste in toys.

One might wonder if a private security or detective agency would have one for rent. They wouldn't need many clients to pay off that investment.

## Spooks may break DES faster yet

As for the security and intelligence agencies of various nations, they may have had DES crackers for years, and theirs may be much faster. It is difficult to make most computer applications work well on parallel

machines, or to design specialised hardware to accelerate them. Cipher-cracking is one of the very few exceptions. It is entirely straightforward to speed up cracking by just adding hardware. Within very broad limits, you can make it as fast as you like if you have the budget. The EFF's \$200,000 machine breaks DES in a few days. An [aviation website](#) gives the cost of a B1 bomber as \$200,000,000. Spending that much, an intelligence agency could break DES in an average time of *six and a half minutes*.

That estimate assumes they use the EFF's 1998 technology and just spend more money. They may have an attack that is superior to brute force, they quite likely have better chip technology (Moore's law, a bigger budget, and whatever secret advances they may have made) and of course they may have spent the price of an aircraft carrier, not just one aircraft.

In short, we have *no idea* how quickly these organisations can break DES. Unless they're spectacularly incompetent or horribly underfunded, they can certainly break it, but we cannot guess how quickly. Pick any time unit between days and milliseconds; none is entirely unbelievable. More to the point, none of them is of any comfort if you don't want such organisations reading your communications.

Note that this may be a concern even if nothing you do is a threat to anyone's national security. An intelligence agency might well consider it to be in their national interest for certain companies to do well. If you're competing against such companies in a world market and that agency can read your secrets, you have a serious problem.

One might wonder about technology the former Soviet Union and its allies developed for cracking DES during the Cold War. They must have tried; the cipher was an American standard and widely used. Certainly those countries have some fine mathematicians, and those agencies had budget. How well did they succeed? Is their technology now for sale or rent?

## Networks break DES in a few weeks

Before the definitive EFF effort, DES had been cracked several times by people using many machines. See [this press release](#) for example.

A major corporation, university, or government department could break DES by using spare cycles on their existing collection of computers, by dedicating a group of otherwise surplus machines to the problem, or by combining the two approaches. It might take them weeks or months, rather than the days required for the EFF machine, but they could do it.

What about someone working alone, without the resources of a large organisation? For them, cracking DES will not be easy, but it may be possible. A few thousand dollars buys a lot of surplus workstations. A pile of such machines will certainly heat your garage nicely and might break DES in a few months or years. Or enroll at a university and use their machines. Or use an employer's machines. Or crack security somewhere and steal the resources to crack a DES key. Or write a virus that steals small amounts of resources on many machines. Or . . .

None of these approaches are easy or break DES really quickly, but an attacker only needs to find one that is feasible and breaks DES quickly enough to be dangerous. How much would you care to bet that this will be impossible if the attacker is clever and determined? How valuable is your data? Are you authorised to risk it on a dubious bet?

## We disable DES

In short, it is now absolutely clear that *DES is not secure* against

- any *well-funded opponent*
- any opponent (even a penniless one) with access (even stolen access) to *enough general purpose computers*

That is why *Linux FreeS/WAN disables all transforms which use plain DES* for encryption.

DES is in the source code, because we need DES to implement our default encryption transform, Triple DES. *We urge you not to use single DES*. We do not provide any easy way to enable it in FreeS/WAN, and our policy is to provide no assistance to anyone wanting to do so.

## 40-bits is laughably weak

The same is true, in spades, of ciphers — DES or others — crippled by 40-bit keys, as many ciphers were required to be until recently under various export laws. A brute force search of such a cipher's key space is  $2^{16}$  times faster than a similar search against DES. The EFF's machine can do a brute-force search of a 40-bit key space in *seconds*. One contest to crack a 40-bit cipher was won by a student using a few hundred idle machines at his university. It took only three and half hours.

We do not, and will not, implement any 40-bit cipher.

## Triple DES is almost certainly secure

Triple DES, usually abbreviated 3DES, applies DES three times, with three different keys. DES seems to be basically an excellent cipher design; it has withstood several decades of intensive analysis without any disastrous flaws being found. It's only major flaw is that the small key space allows brute force attacks to succeed. Triple DES enlarges the key space to 168 bits, making brute-force search a ridiculous impossibility.

3DES is currently the only block cipher implemented in FreeS/WAN. 3DES is, unfortunately, about 1/3 the speed of DES, but modern CPUs still do it at quite respectable speeds. Some speed measurements for our code are available.

## AES in IPsec

The AES project has chosen a replacement for DES, a new standard cipher for use in non-classified US government work and in regulated industries such as banking. This cipher will almost certainly become widely used for many applications, including IPsec.

The winner, announced in October 2000 after several years of analysis and discussion, was the Rijndael cipher from two Belgian designers.

It is almost certain that FreeS/WAN will add AES support. AES patches are already available.

## Press coverage of Linux FreeS/WAN:

### FreeS/WAN 1.0 press

- [Wired](#) "Linux-Based Crypto Stops Snoops", James Glave April 15 1999
- [Slashdot](#)
- [DGL](#), Damar Group Limited; looking at FreeS/WAN from a perspective of business computing
- [Linux Today](#)
- [TBTE](#), Tasty Bits from the Technology Front
- [Salon Magazine](#) "Free Encryption Takes a Big Step"

### Press release for version 1.0

Strong Internet Privacy Software Free for Linux Users Worldwide

Toronto, ON, April 14, 1999 -

The Linux FreeS/WAN project today released free software to protect the privacy of Internet communications using strong encryption codes. FreeS/WAN automatically encrypts data as it crosses the Internet, to prevent unauthorized people from receiving or modifying it. One ordinary PC per site runs this free software under Linux to become a secure gateway in a Virtual Private Network, without having to modify users' operating systems or application software. The project built and released the software outside the United States, avoiding US government regulations which prohibit good privacy protection. FreeS/WAN version 1.0 is available immediately for downloading at <http://www.xs4all.nl/~freeswan/>.

"Today's FreeS/WAN release allows network administrators to build excellent secure gateways out of old PCs at no cost, or using a cheap new PC," said John Gilmore, the entrepreneur who instigated the project in 1996. "They can build operational experience with strong network encryption and protect their users' most important communications worldwide."

"The software was written outside the United States, and we do not accept contributions from US citizens or residents, so that it can be freely published for use in every country," said Henry Spencer, who built the release in Toronto, Canada. "Similar products based in the US require hard-to-get government export licenses before they can be provided to non-US users, and can never be simply published on a Web site. Our product is freely available worldwide for immediate downloading, at no cost."

FreeS/WAN provides privacy against both quiet eavesdropping (such as "packet sniffing") and active attempts to compromise communications (such as impersonating participating computers). Secure "tunnels" carry information safely across the Internet between locations such as a company's main office, distant sales offices, and roaming laptops. This protects the privacy and integrity of all information sent among those locations, including sensitive intra-company email, financial transactions such as mergers and acquisitions, business negotiations, personal medical records, privileged correspondence with lawyers, and information about crimes or civil rights violations. The software will be particularly useful to frequent wiretapping targets such as private companies competing with government-owned companies, civil rights groups and lawyers, opposition political parties, and dissidents.

## Introduction to FreeS/WAN

FreeS/WAN provides privacy for Internet packets using the proposed standard Internet Protocol Security (IPSEC) protocols. FreeS/WAN negotiates strong keys using Diffie-Hellman key agreement with 1024-bit keys, and encrypts each packet with 168-bit Triple-DES (3DES). A modern \$500 PC can set up a tunnel in less than a second, and can encrypt 6 megabits of packets per second, easily handling the whole available bandwidth at the vast majority of Internet sites. In preliminary testing, FreeS/WAN interoperated with 3DES IPSEC products from OpenBSD, PGP, SSH, Cisco, Raptor, and Xedia. Since FreeS/WAN is distributed as source code, its innards are open to review by outside experts and sophisticated users, reducing the chance of undetected bugs or hidden security compromises.

The software has been in development for several years. It has been funded by several philanthropists interested in increased privacy on the Internet, including John Gilmore, co-founder of the Electronic Frontier Foundation, a leading online civil rights group.

Press contacts:

Hugh Daniel, +1 408 353 8124, [hugh@toad.com](mailto:hugh@toad.com)

Henry Spencer, +1 416 690 6561, [henry@spsystems.net](mailto:henry@spsystems.net)

\* FreeS/WAN derives its name from S/WAN, which is a trademark of RSA Data Security, Inc; used by permission.

# The IPsec protocols

This section provides information on the IPsec protocols which FreeS/WAN implements. For more detail, see the RFCs.

The basic idea of IPsec is to provide security functions, authentication and encryption, at the IP (Internet Protocol) level. This requires a higher-level protocol (IKE) to set things up for the IP-level services (ESP and AH).

## Protocols and phases

Three protocols are used in an IPsec implementation:

*ESP, Encapsulating Security Payload*

Encrypts and/or authenticates data

*AH, Authentication Header*

Provides a packet authentication service

*IKE, Internet Key Exchange*

Negotiates connection parameters, including keys, for the other two

The term "IPsec" (also written as IPSEC) is slightly ambiguous. In some contexts, it includes all three of the above but in other contexts it refers only to AH and ESP.

There is more detail below, but a quick summary of how the whole thing works is:

*Phase one IKE (main mode exchange)*

sets up a keying channel (ISAKMP SA) between the two gateways

*Phase two IKE (quick mode exchange)*

sets up data channels (IPsec SAs)

*IPsec proper*

exchanges data using AH or ESP

Both phases of IKE are repeated periodically to automate re-keying.

## Applying IPsec

Authentication and encryption functions for network data can, of course, be provided at other levels. Many security protocols work at levels above IP.

- PGP encrypts and authenticates mail messages
- SSH authenticates remote logins and then encrypts the session
- SSL or TLS provides security at the sockets layer, e.g. for secure web browsing

and so on. Other techniques work at levels below IP. For example, data on a communications circuit or an entire network can be encrypted by specialised hardware. This is common practice in high-security applications.

## Advantages of IPsec

There are, however, advantages to doing it at the IP level instead of, or as well as, at other levels.

IPsec is the *most general way to provide these services for the Internet*.

- Higher-level services protect a *single protocol*; for example PGP protects mail.
- Lower level services protect a *single medium*; for example a pair of encryption boxes on the ends of a line make wiretaps on that line useless unless the attacker is capable of breaking the encryption.

IPsec, however, can protect *any protocol* running above IP and *any medium* which IP runs over. More to the point, it can protect a mixture of application protocols running over a complex combination of media. This is the normal situation for Internet communication; IPsec is the only general solution.

IPsec can also provide some security services "in the background", with *no visible impact on users*. To use PGP encryption and signatures on mail, for example, the user must at least:

- remember his or her passphrase,
- keep it secure
- follow procedures to validate correspondents' keys

These systems can be designed so that the burden on users is not onerous, but any system will place some requirements on users. No such system can hope to be secure if users are sloppy about meeting those requirements. The author has seen username and password stuck on terminals with post-it notes in an allegedly secure environment, for example.

## Limitations of IPsec

IPsec is designed to secure IP links between machines. It does that well, but it is important to remember that there are many things it does not do. Some of the important limitations are:

*IPsec cannot be secure if your system isn't*

System security on IPsec gateway machines is an essential requirement if IPsec is to function as designed. No system can be trusted if the underlying machine has been subverted. See books on Unix security such as Garfinkel and Spafford or our web references for Linux security or more general computer security.

Of course, there is another side to this. IPsec can be a powerful tool for improving system and network security. For example, requiring packet authentication makes various spoofing attacks harder and IPsec tunnels can be extremely useful for secure remote administration of various things.

*IPsec is not end-to-end*

IPsec cannot provide the same end-to-end security as systems working at higher levels. IPsec encrypts an IP connection between two machines, which is quite a different thing than encrypting messages between users or between applications.

For example, if you need mail encrypted from the sender's desktop to the recipient's desktop and decryptable only by the recipient, use PGP or another such system. IPsec can encrypt any or all of the links involved — between the two mail servers, or between either server and its clients. It could even be used to secure a direct IP link from the sender's desktop machine to the recipient's, cutting out any sort of network snoop. What it cannot ensure is end-to-end user-to-user security. If only IPsec is

used to secure mail, then anyone with appropriate privileges on any machine where that mail is stored (at either end or on any store-and-forward servers in the path) can read it.

In another common setup, IPsec encrypts packets at a security gateway machine as they leave the sender's site and decrypts them on arrival at the gateway to the recipient's site. This does provide a useful security service — only encrypted data is passed over the Internet — but it does not even come close to providing an end-to-end service. In particular, anyone with appropriate privileges on either site's LAN can intercept the message in unencrypted form.

### *IPsec cannot do everything*

IPsec also cannot provide all the functions of systems working at higher levels of the protocol stack. If you need a document electronically signed by a particular person, then you need his or her digital signature and a public key cryptosystem to verify it with.

Note, however, that IPsec authentication of the underlying communication can make various attacks on higher-level protocols more difficult. In particular, authentication prevents man-in-the-middle attacks.

### *IPsec authenticates machines, not users*

IPsec uses strong authentication mechanisms to control which messages go to which machines, but it does not have the concept of user ID, which is vital to many other security mechanisms and policies. This means some care must be taken in fitting the various security mechanisms on a network together. For example, if you need to control which users access your database server, you need some non-IPsec mechanism for that. IPsec can control which machines connect to the server, and can ensure that data transfer to those machines is done securely, but that is all. Either the machines themselves must control user access or there must be some form of user authentication to the database, independent of IPsec.

### *IPsec does not stop denial of service attacks*

Denial of service attacks aim at causing a system to crash, overload, or become confused so that legitimate users cannot get whatever services the system is supposed to provide. These are quite different from attacks in which the attacker seeks either to use the service himself or to subvert the service into delivering incorrect results.

IPsec shifts the ground for DoS attacks; the attacks possible against systems using IPsec are different than those that might be used against other systems. It does not, however, eliminate the possibility of such attacks.

### *IPsec does not stop traffic analysis*

Traffic analysis is the attempt to derive intelligence from messages without regard for their contents. In the case of IPsec, it would mean analysis based on things visible in the unencrypted headers of encrypted packets — source and destination gateway addresses, packet size, et cetera. Given the resources to acquire such data and some skill in analysing it (both of which any national intelligence agency should have), this can be a very powerful technique.

IPsec is not designed to defend against this. Partial defenses are certainly possible, and some are described below, but it is not clear that any complete defense can be provided.

## **IPsec is a general mechanism for securing IP**

While IPsec does not provide all functions of a mail encryption package, it can encrypt your mail. In particular, it can ensure that all mail passing between a pair or a group of sites is encrypted. An attacker looking only at external traffic, without access to anything on or behind the IPsec gateway, cannot read your mail. He or she is stymied by IPsec just as he or she would be by PGP.

The advantage is that IPsec can provide the same protection for *anything transmitted over IP*. In a corporate network example, PGP lets the branch offices exchange secure mail with head office. SSL and SSH allow them to securely view web pages, connect as terminals to machines, and so on. IPsec can support all those applications, plus database queries, file sharing (NFS or Windows), other protocols encapsulated in IP (Netware, Appletalk, ...), phone-over-IP, video-over-IP, ... anything-over-IP. The only limitation is that IP Multicast is not yet supported, though there are Internet Draft documents for that.

IPsec creates *secure tunnels through untrusted networks*. Sites connected by these tunnels form VPNs, Virtual Private Networks.

IPsec gateways can be installed wherever they are required.

- One organisation might choose to install IPsec only on firewalls between their LANs and the Internet. This would allow them to create a VPN linking several offices. It would provide protection against anyone outside their sites.
- Another might install IPsec on departmental servers so everything on the corporate backbone net was encrypted. This would protect messages on that net from everyone except the sending and receiving department.
- Another might be less concerned with information secrecy and more with controlling access to certain resources. They might use IPsec packet authentication as part of an access control mechanism, with or without also using the IPsec encryption service.
- It is even possible (assuming adequate processing power and an IPsec implementation in each node) to make every machine its own IPsec gateway so that everything on a LAN is encrypted. This protects information from everyone outside the sending and receiving machine.
- These techniques can be combined in various ways. One might, for example, require authentication everywhere on a network while using encryption only for a few links.

Which of these, or of the many other possible variants, to use is up to you. *IPsec provides mechanisms; you provide the policy.*

*No end user action is required* for IPsec security to be used; they don't even have to know about it. The site administrators, of course, do have to know about it and to put some effort into making it work. Poor administration can compromise IPsec as badly as the post-it notes mentioned above. It seems reasonable, though, for organisations to hope their system administrators are generally both more security-conscious than end users and more able to follow computer security procedures. If not, at least there are fewer of them to educate or replace.

IPsec can be, and often should be, used with along with security protocols at other levels. If two sites communicate with each other via the Internet, then IPsec is the obvious way to protect that communication. If two others have a direct link between them, either link encryption or IPsec would make sense. Choose one or use both. Whatever you use at and below the IP level, use other things as required above that level. Whatever you use above the IP level, consider what can be done with IPsec to make attacks on the higher levels harder. For example, man-in-the-middle attacks on various protocols become difficult if authentication at packet level is in use on the potential victims' communication channel.

## Using authentication without encryption

Where appropriate, IPsec can provide authentication without encryption. One might do this, for example:

- where the data is public but one wants to be sure of getting the right data, for example on some web

sites

- where encryption is judged unnecessary, for example on some company or department LANs
- where strong encryption is provided at link level, below IP
- where strong encryption is provided in other protocols, above IP

Note that IPsec authentication may make some attacks on those protocols harder.

Authentication has lower overheads than encryption.

The protocols provide four ways to build such connections, using either an AH-only connection or ESP using null encryption, and in either manually or automatically keyed mode. FreeS/WAN supports only one of these, manually keyed AH-only connections, and ***we do not recommend using that***. Our reasons are discussed under Resisting traffic analysis a few sections further along.

## Encryption without authentication is dangerous

Originally, the IPsec encryption protocol ESP didn't do integrity checking. It only did encryption. Steve Bellovin found many ways to attack ESP used without authentication. See his paper Problem areas for the IP Security Protocols. To make a secure connection, you had to add an AH Authentication Header as well as ESP. Rather than incur the overhead of several layers (and rather than provide an ESP layer that didn't actually protect the traffic), the IPsec working group built integrity and replay checking directly into ESP.

Today, typical usage is one of:

- ESP for encryption and authentication
- AH for authentication alone

Other variants are allowed by the standard, but not much used:

*ESP encryption without authentication*

***Bellovin has demonstrated fatal flaws in this. Do not use.***

*ESP encryption with AH authentication*

This has higher overheads than using the authentication in ESP, and no obvious benefit in most cases. The exception might be a network where AH authentication was widely or universally used. If you're going to do AH to conform with network policy, why authenticate again in the ESP layer?

*Authenticate twice, with AH and with ESP*

Why? Of course, some folk consider "belt and suspenders" the sensible approach to security. If you're among them, you might use both protocols here. You might also use both to satisfy different parts of a security policy. For example, an organisation might require AH authentication everywhere but two users within the organisation might use ESP as well.

*ESP authentication without encryption*

The standard allows this, calling it "null encryption". FreeS/WAN does not support it. We recommend that you use AH instead if authentication is all you require. AH authenticates parts of the IP header, which ESP-null does not do.

Some of these variants cannot be used with FreeS/WAN because we do not support ESP-null and do not support automatic keying of AH-only connections.

There are fairly frequent suggestions that AH be dropped entirely from the IPsec specifications since ESP and null encryption can handle that situation. It is not clear whether this will occur. My guess is that it is unlikely.

## Multiple layers of IPsec processing are possible

The above describes combinations possible on a single IPsec connection. In a complex network you may have several layers of IPsec in play, with any of the above combinations at each layer.

For example, a connection from a desktop machine to a database server might require AH authentication. Working with other host, network and database security measures, AH might be just the thing for access control. You might decide not to use ESP encryption on such packets, since it uses resources and might complicate network debugging. Within the site where the server is, then, only AH would be used on those packets.

Users at another office, however, might have their whole connection (AH headers and all) passing over an IPsec tunnel connecting their office to the one with the database server. Such a tunnel should use ESP encryption and authentication. You need authentication in this layer because without authentication the encryption is vulnerable and the gateway cannot verify the AH authentication. The AH is between client and database server; the gateways aren't party to it.

In this situation, some packets would get multiple layers of IPsec applied to them, AH on an end-to-end client-to-server basis and ESP from one office's security gateway to the other.

## Resisting traffic analysis

Traffic analysis is the attempt to derive useful intelligence from encrypted traffic without breaking the encryption.

Is your CEO exchanging email with a venture capital firm? With bankruptcy trustees? With an executive recruiting agency? With the holder of some important patents? If an eavesdropper learns about any of those, then he has interesting intelligence on your company, whether or not he can read the messages themselves.

Even just knowing that there is network traffic between two sites may tell an analyst something useful, especially when combined with whatever other information he or she may have. For example, if you know Company A is having cashflow problems and Company B is looking for acquisitions, then knowing that packets are passing between the two is interesting. It is more interesting if you can tell it is email, and perhaps yet more if you know the sender and recipient.

Except in the simplest cases, traffic analysis is hard to do well. It requires both considerable resources and considerable analytic skill. However, intelligence agencies of various nations have been doing it for centuries and many of them are likely quite good at it by now. Various commercial organisations, especially those working on "targeted marketing" may also be quite good at analysing certain types of traffic.

In general, defending against traffic analysis is also difficult. Inventing a really good defense could get you a PhD and some interesting job offers.

IPsec is not designed to stop traffic analysis and we know of no plausible method of extending it to do so. That said, there are ways to make traffic analysis harder. This section describes them.

### Using "unnecessary" encryption

One might choose to use encryption even where it appears unnecessary in order to make analysis more difficult. Consider two offices which pass a small volume of business data between them using IPsec and also

transfer large volumes of Usenet news. At first glance, it would seem silly to encrypt the newsfeed, except possibly for any newsgroups that are internal to the company. Why encrypt data that is all publicly available from many sites?

However, if we encrypt a lot of news and send it down the same connection as our business data, we make traffic analysis much harder. A snoop cannot now make inferences based on patterns in the volume, direction, sizes, sender, destination, or timing of our business messages. Those messages are hidden in a mass of news messages encapsulated in the same way.

If we're going to do this we need to ensure that keys change often enough to remain secure even with high volumes and with the adversary able to get plaintext of much of the data. We also need to look at other attacks this might open up. For example, can the adversary use a chosen plaintext attack, deliberately posting news articles which, when we receive and encrypt them, will help break our encryption? Or can he block our business data transmission by flooding us with silly news articles? Or ...

Also, note that this does not provide complete protection against traffic analysis. A clever adversary might still deduce useful intelligence from statistical analysis (perhaps comparing the input newsfeed to encrypted output, or comparing the streams we send to different branch offices), or by looking for small packets which might indicate establishment of TCP connections, or ...

As a general rule, though, to improve resistance to traffic analysis, you should ***encrypt as much traffic as possible, not just as much as seems necessary.***

### Using multiple encryption

This also applies to using multiple layers of encryption. If you have an IPsec tunnel between two branch offices, it might appear silly to send PGP-encrypted email through that tunnel. However, if you suspect someone is snooping your traffic, then it does make sense:

- it protects the mail headers; they cannot even see who is mailing who
- it protects against user bumbles or software malfunctions that accidentally send messages in the clear
- it makes any attack on the mail encryption much harder; they have to break IPsec or break into your network before they can start on the mail encryption

Similar arguments apply for SSL-encrypted web traffic or SSH-encrypted remote login sessions, even for end-to-end IPsec tunnels between systems in the two offices.

### Using fewer tunnels

It may also help to use fewer tunnels. For example, if all you actually need encrypted is connections between:

- mail servers at branch and head offices
- a few branch office users and the head office database server

You might build one tunnel per mail server and one per remote database user, restricting traffic to those applications. This gives the traffic analyst some information, however. He or she can distinguish the tunnels by looking at information in the ESP header and, given that distinction and the patterns of tunnel usage, might be able to figure out something useful. Perhaps not, but why take the risk?

We suggest instead that you build one tunnel per branch office, encrypting everything passing from head office to branches. This has a number of advantages:

- it is easier to build and administer
- it resists traffic analysis somewhat better
- it provides security for whatever you forgot. For example, if some user at a remote office browses proprietary company data on some head office web page (that the security people may not even know about!), then that data is encrypted before it reaches the Internet.

Of course you might also want to add additional tunnels. For example, if some of the database data is confidential and should not be exposed even within the company, then you need protection from the user's desktop to the database server. We suggest you do that in whatever way seems appropriate — IPsec, SSH or SSL might fit — but, whatever you choose, pass it between locations via a gateway-to-gateway IPsec tunnel to provide some resistance to traffic analysis.

## Cryptographic components

IPsec combines a number of cryptographic techniques, all of them well-known and well-analyzed. The overall design approach was conservative; no new or poorly-understood components were included.

This section gives a brief overview of each technique. It is intended only as an introduction. There is more information, and links to related topics, in our [glossary](#). See also our [bibliography](#) and cryptography [web links](#).

### Block ciphers

The [encryption](#) in the [ESP](#) encapsulation protocol is done with a [block cipher](#).

We do not implement [single DES](#). It is [insecure](#). Our default, and currently only, block cipher is [triple DES](#).

The [Rijndael](#) block cipher has won the [AES](#) competition to choose a replacement for DES. It will almost certainly be added to FreeS/WAN and to other IPsec implementations. [Patches](#) are already available.

### Hash functions

#### The HMAC construct

IPsec packet authentication is done with the [HMAC](#) construct. This is not just a hash of the packet data, but a more complex operation which uses both a hashing algorithm and a key. It therefore does more than a simple hash would. A simple hash would only tell you that the packet data was not changed in transit, or that whoever changed it also regenerated the hash. An HMAC also tells you that the sender knew the HMAC key.

For IPsec HMAC, the output of the hash algorithm is truncated to 96 bits. This saves some space in the packets. More important, it prevents an attacker from seeing all the hash output bits and perhaps creating some sort of attack based on that knowledge.

#### Choice of hash algorithm

The IPsec RFCs require two hash algorithms — [MD5](#) and [SHA-1](#) — both of which FreeS/WAN implements.

Various other algorithms — such as RIPEMD and Tiger — are listed in the RFCs as optional. None of these are in the FreeS/WAN distribution, or are likely to be added, although user [patches](#) exist for several of them.

Additional hash algorithms — SHA-256, SHA-384 and SHA-512 — may be required to give hash strength matching the strength of AES. These are likely to be added to FreeS/WAN along with AES.

## Diffie–Hellman key agreement

The Diffie–Hellman key agreement protocol allows two parties (A and B or Alice and Bob) to agree on a key in such a way that an eavesdropper who intercepts the entire conversation cannot learn the key.

The protocol is based on the discrete logarithm problem and is therefore thought to be secure. Mathematicians have been working on that problem for years and seem no closer to a solution, though there is no proof that an efficient solution is impossible.

## RSA authentication

The RSA algorithm (named for its inventors — Rivest, Shamir and Adleman) is a very widely used public key cryptographic technique. It is used in IPsec as one method of authenticating gateways for Diffie–Hellman key negotiation.

## Structure of IPsec

There are three protocols used in an IPsec implementation:

*ESP, Encapsulating Security Payload*

Encrypts and/or authenticates data

*AH, Authentication Header*

Provides a packet authentication service

*IKE, Internet Key Exchange*

Negotiates connection parameters, including keys, for the other two

The term "IPsec" is slightly ambiguous. In some contexts, it includes all three of the above but in other contexts it refers only to AH and ESP.

## IKE (Internet Key Exchange)

The IKE protocol sets up IPsec (ESP or AH) connections after negotiating appropriate parameters (algorithms to be used, keys, connection lifetimes) for them. This is done by exchanging packets on UDP port 500 between the two gateways.

IKE (RFC 2409) was the outcome of a long, complex process in which quite a number of protocols were proposed and debated. Oversimplifying mildly, IKE combines:

*ISAKMP (RFC 2408)*

The **I**nternet **S**ecurity Association and **K**ey **M**anagement **P**rotocol manages negotiation of connections and defines SAs (Security Associations) as a means of describing connection properties.

*IPsec DOI for ISAKMP (RFC 2407)*

A **D**omain **O**f **I**nterpretation fills in the details necessary to turn the rather abstract ISAKMP protocol into a more tightly specified protocol, so it becomes applicable in a particular domain.

*Oakley key determination protocol (RFC 2412)*

Oakley creates keys using the Diffie–Hellman key agreement protocol.

For all the details, you would need to read the four RFCs just mentioned (over 200 pages) and a number of others. We give a summary below, but it is far from complete.

### Phases of IKE

IKE negotiations have two phases.

#### *Phase one*

The two gateways negotiate and set up a two-way ISAKMP SA which they can then use to handle phase two negotiations. One such SA between a pair of gateways can handle negotiations for multiple tunnels.

#### *Phase two*

Using the ISAKMP SA, the gateways negotiate IPsec (ESP and/or AH) SAs as required. IPsec SAs are unidirectional (a different key is used in each direction) and are always negotiated in pairs to handle two-way traffic. There may be more than one pair defined between two gateways.

Both of these phases use the UDP protocol and port 500 for their negotiations.

After both IKE phases are complete, you have IPsec SAs to carry your encrypted data. These use the ESP or AH protocols. These protocols do not have ports. Ports apply only to UDP or TCP.

The IKE protocol is designed to be extremely flexible. Among the things that can be negotiated (separately for each SA) are:

- SA lifetime before rekeying
- encryption algorithm used. We currently support only triple DES. Single DES is insecure. The RFCs say you MUST do DES, SHOULD do 3DES and MAY do various others. We do not do any of the others.
- authentication algorithms. We support MD5 and SHA. These are the two the RFCs require.
- choice of group for Diffie–Hellman key agreement. We currently support Groups 2 and 5 (which are defined modulo primes of various lengths) and do not support Group 1 (defined modulo a shorter prime, and therefore cryptographically weak) or groups 3 and 4 (defined using elliptic curves). The RFCs require only Group 1.

The protocol also allows implementations to add their own encryption algorithms, authentication algorithms or Diffie–Hellman groups. We do not support any such extensions, but there are some patches that do.

There are a number of complications:

- The gateways must be able to authenticate each other's identities before they can create a secure connection. This host authentication is part of phase one negotiations, and is a required prerequisite for packet authentication used later. Host authentication can be done in a variety of ways. Those supported by FreeS/WAN are discussed in our advanced configuration document.
- Phase one can be done in two ways.
  - ◆ Main Mode is required by the RFCs and supported in FreeS/WAN. It uses a 6–packet exchange.
  - ◆ Aggressive Mode is somewhat faster (only 3 packets) but reveals more to an eavesdropper. This is optional in the RFCs, not currently supported by FreeS/WAN, and not likely to be.
- A new group exchange may take place after phase one but before phase two, defining an additional group for use in the Diffie–Hellman key agreement part of phase two. FreeS/WAN does not currently support this.

## Introduction to FreeS/WAN

- Phase two always uses Quick Mode, but there are two variants of that:
  - ◆ One variant provides Perfect Forward Secrecy (PFS). An attacker that obtains your long-term host authentication key does not immediately get any of your short-term packet encryption or packet authentication keys. He must conduct another successful attack each time you rekey to get the short-term keys. Having some short-term keys does not help him learn others. In particular, breaking your system today does not let him read messages he archived yesterday, assuming you've changed short-term keys in the meanwhile. We enable PFS as the default.
  - ◆ The other variant disables PFS and is therefore slightly faster. We do not recommend this since it is less secure, but FreeS/WAN does support it. You can enable it with a *pfs=no* statement in ipsec.conf(5).
  - ◆ The protocol provides no way to negotiate which variant will be used. If one gateway is set for PFS and the other is not, the negotiation fails. This has proved a fairly common source of interoperation problems.
- Several types of notification message may be sent by either side during either phase, or later. FreeS/WAN does not currently support these, but they are a likely addition in future releases.
- There is a commit flag which may optionally be set on some messages. The errata page for the RFCs includes two changes related to this, one to clarify the description of its use and one to block a denial of service attack which uses it. We currently do not implement this feature.

These complications can of course lead to problems, particularly when two different implementations attempt to interoperate. For example, we have seen problems such as:

- Some implementations (often products crippled by export laws) have the insecure DES algorithm as their only supported encryption method. Other parts of our documentation discuss the reasons we do not implement single DES, and how to cope with crippled products.
- Windows 2000 IPsec tries to negotiate using Aggressive Mode, which we don't support. Later on, it uses the commit bit, which we also don't support.
- Various implementations disable PFS by default, and therefore will not talk to FreeS/WAN until you either turn on PFS on their end or turn it off in FreeS/WAN with a *pfs=no* entry in the connection description.
- FreeS/WAN's interaction with PGPnet is complicated by their use of notification messages we do not yet support.

Despite this, we do interoperate successfully with many implementations, including both Windows 2000 and PGPnet. Details are in our interoperability document.

### Sequence of messages in IKE

Each phase (see previous section) of IKE involves a series of messages. In Pluto error messages, these are abbreviated using:

<i>M</i>	<i>Main mode</i> , setting up the keying channel (ISAKMP SA)
<i>Q</i>	<i>Quick mode</i> , setting up the data channel (IPsec SA)
<i>I</i>	<i>Initiator</i> , the machine that starts the negotiation
<i>R</i>	<i>Responder</i>

For example, the six messages of a main mode negotiation, in sequence, are labelled:

```
MI1 ----->
    <----- MR1
MI2 ----->
    <----- MR2
MI3 ----->
    <----- MR3
```

### Structure of IKE messages

Here is our Pluto developer explaining some of this on the mailing list:

When one IKE system (for example, Pluto) is negotiating with another to create an SA, the Initiator proposes a bunch of choices and the Responder replies with one that it has selected.

The structure of the choices is fairly complicated. An SA payload contains a list of lists of "Proposals". The outer list is a set of choices: the selection must be from one element of this list.

Each of these elements is a list of Proposals. A selection must be made from each of the elements of the inner list. In other words, *\*all\** of them apply (that is how, for example, both AH and ESP can apply at once).

Within each of these Proposals is a list of Transforms. For each Proposal selected, one Transform must be selected (in other words, each Proposal provides a choice of Transforms).

Each Transform is made up of a list of Attributes describing, well, attributes. Such as lifetime of the SA. Such as algorithm to be used. All the Attributes apply to a Transform.

You will have noticed a pattern here: layers alternate between being disjunctions ("or") and conjunctions ("and").

For Phase 1 / Main Mode (negotiating an ISAKMP SA), this structure is cut back. There must be exactly one Proposal. So this degenerates to a list of Transforms, one of which must be chosen.

### IPsec Services, AH and ESP

IPsec offers two services, authentication and encryption. These can be used separately but are often used together.

#### *Authentication*

Packet-level authentication allows you to be confident that a packet came from a particular machine and that its contents were not altered en route to you. No attempt is made to conceal or protect the contents, only to assure their integrity. Packet authentication can be provided separately using an Authentication Header, described just below, or it can be included as part of the ESP (Encapsulated Security Payload) service, described in the following section. That service offers encryption as well as authentication. In either case, the HMAC construct is used as the authentication mechanism.

There is a separate authentication operation at the IKE level, in which each gateway authenticates the other. This can be done in a variety of ways.

#### *Encryption*

Encryption allows you to conceal the contents of a message from eavesdroppers.

In IPsec this is done using a block cipher (normally Triple DES for Linux). In the most used setup, keys are automatically negotiated, and periodically re-negotiated, using the IKE (Internet Key Exchange) protocol. In Linux FreeS/WAN this is handled by the Pluto Daemon.

The IPsec protocol offering encryption is ESP, Encapsulated Security Payload. It can also include a packet authentication service.

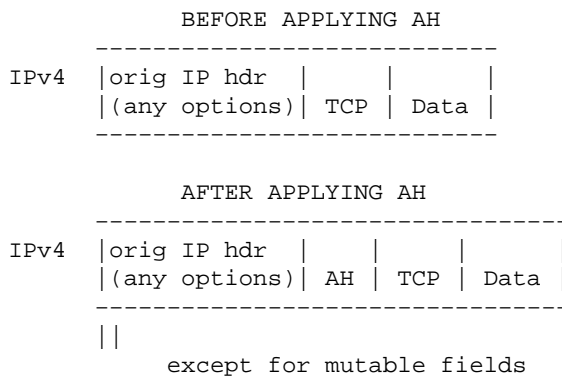
Note that *encryption should always be used with some packet authentication service*. Unauthenticated encryption is vulnerable to man-in-the-middle attacks. Also note that encryption does not prevent traffic analysis.

## The Authentication Header (AH)

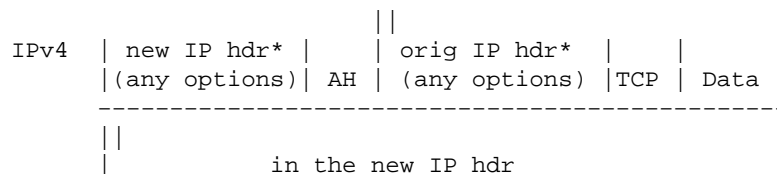
Packet authentication can be provided separately from encryption by adding an authentication header (AH) after the IP header but before the other headers on the packet. This is the subject of this section. Details are in RFC 2402.

Each of the several headers on a packet header contains a "next protocol" field telling the system what header to look for next. IP headers generally have either TCP or UDP in this field. When IPsec authentication is used, the packet IP header has AH in this field, saying that an Authentication Header comes next. The AH header then has the next header type — usually TCP, UDP or encapsulated IP.

IPsec packet authentication can be added in transport mode, as a modification of standard IP transport. This is shown in this diagram from the RFC:



Athentication can also be used in tunnel mode, encapsulating the underlying IP packet beneath AH and an additional IP header.



This would normally be used in a gateway-to-gateway tunnel. The receiving gateway then strips the outer IP header and the AH header and forwards the inner IP packet.

The mutable fields referred to are things like the time-to-live field in the IP header. These cannot be included in authentication calculations because they change as the packet travels.

### Keyed MD5 and Keyed SHA

The actual authentication data in the header is typically 96 bits and depends both on a secret shared between sender and receiver and on every byte of the data being authenticated. The technique used is HMAC, defined in RFC 2104.

The algorithms involved are the MD5 Message Digest Algorithm or SHA, the Secure Hash Algorithm. For details on their use in this application, see RFCs 2403 and 2404 respectively.

For descriptions of the algorithms themselves, see RFC 1321 for MD5 and FIPS (Federal Information Processing Standard) number 186 from NIST, the US National Institute of Standards and Technology for SHA. Applied Cryptography covers both in some detail, MD5 starting on page 436 and SHA on 442.

These algorithms are intended to make it nearly impossible for anyone to alter the authenticated data in transit. The sender calculates a digest or hash value from that data and includes the result in the authentication header. The recipient does the same calculation and compares results. For unchanged data, the results will be identical. The hash algorithms are designed to make it extremely difficult to change the data in any way and still get the correct hash.

Since the shared secret key is also used in both calculations, an interceptor cannot simply alter the authenticated data and change the hash value to match. Without the key, he or she (or even the dreaded They) cannot produce a usable hash.

### Sequence numbers

The authentication header includes a sequence number field which the sender is required to increment for each packet. The receiver can ignore it or use it to check that packets are indeed arriving in the expected sequence.

This provides partial protection against replay attacks in which an attacker resends intercepted packets in an effort to confuse or subvert the receiver. Complete protection is not possible since it is necessary to handle legitimate packets which are lost, duplicated, or delivered out of order, but use of sequence numbers makes the attack much more difficult.

The RFCs require that sequence numbers never cycle, that a new key always be negotiated before the sequence number reaches  $2^{32}-1$ . This protects both against replays attacks using packets from a previous cycle and against birthday attacks on the the packet authentication algorithm.

In Linux FreeS/WAN, the sequence number is ignored for manually keyed connections and checked for automatically keyed ones. In manual mode, there is no way to negotiate a new key, or to recover from a sequence number problem, so we don't use sequence numbers.

### Encapsulated Security Payload (ESP)

The ESP protocol is defined in RFC 2406. It provides one or both of encryption and packet authentication. It may be used with or without AH packet authentication.

Note that *some form of packet authentication should always be used whenever data is encrypted*. Without authentication, the encryption is vulnerable to active attacks which may allow an enemy to break the encryption. ESP should *always* either include its own authentication or be used with AH authentication.

The RFCs require support for only two mandatory encryption algorithms — DES, and null encryption — and for two authentication methods — keyed MD5 and keyed SHA. Implementers may choose to support additional algorithms in either category.

The authentication algorithms are the same ones used in the IPsec authentication header.

We do not implement single DES since DES is insecure. Instead we provide triple DES or 3DES. This is currently the only encryption algorithm supported.

We do not implement null encryption since it is obviously insecure.

## IPsec modes

IPsec can connect in two modes. Transport mode is a host-to-host connection involving only two machines. In tunnel mode, the IPsec machines act as gateways and traffic for any number of client machines may be carried.

### Tunnel mode

Security gateways are required to support tunnel mode connections. In this mode the gateways provide tunnels for use by client machines behind the gateways. The client machines need not do any IPsec processing; all they have to do is route things to gateways.

### Transport mode

Host machines (as opposed to security gateways) with IPsec implementations must also support transport mode. In this mode, the host does its own IPsec processing and routes some packets via IPsec.

## FreeS/WAN parts

### KLIPS: Kernel IPsec Support

KLIPS is *Kernel IPSEC Support*, the modifications necessary to support IPsec within the Linux kernel. KLIPS does all the actual IPsec packet-handling, including

- encryption
- packet authentication calculations
- creation of ESP and AH headers for outgoing packets
- interpretation of those headers on incoming packets

KLIPS also checks all non-IPsec packets to ensure they are not bypassing IPsec security policies.

### The Pluto daemon

Pluto(8) is a daemon which implements the IKE protocol. It

- handles all the Phase one ISAKMP SAs
- performs host authentication and negotiates with other gateways
- creates IPsec SAs and passes the data required to run them to KLIPS

- adjust routing and firewall setup to meet IPsec requirements. See our [IPsec and firewalling](#) document for details.

Pluto is controlled mainly by the [ipsec.conf\(5\)](#) configuration file.

## The ipsec(8) command

The [ipsec\(8\)](#) command is a front end shellsript that allows control over IPsec activity.

## Linux FreeS/WAN configuration file

The configuration file for Linux FreeS/WAN is

```
/etc/ipsec.conf
```

For details see the [ipsec.conf\(5\)](#) manual page .

## Key management

There are several ways IPsec can manage keys. Not all are implemented in Linux FreeS/WAN.

## Currently Implemented Methods

### Manual keying

IPsec allows keys to be manually set. In Linux FreeS/WAN, such keys are stored with the connection definitions in `/etc/ipsec.conf`.

[Manual keying](#) is useful for debugging since it allows you to test the [KLIPS](#) kernel IPsec code without the [Pluto](#) daemon doing key negotiation.

In general, however, automatic keying is preferred because it is more secure.

### Automatic keying

In automatic keying, the [Pluto](#) daemon negotiates keys using the [IKE](#) Internet Key Exchange protocol. Connections are automatically re-keyed periodically.

This is considerably more secure than manual keying. In either case an attacker who acquires a key can read every message encrypted with that key, but automatic keys can be changed every few hours or even every few minutes without breaking the connection or requiring intervention by the system administrators. Manual keys can only be changed manually; you need to shut down the connection and have the two admins make changes. Moreover, they have to communicate the new keys securely, perhaps with [PGP](#) or [SSH](#). This may be possible in some cases, but as a general solution it is expensive, bothersome and unreliable. Far better to let [Pluto](#) handle these chores; no doubt the administrators have enough to do.

Also, automatic keying is inherently more secure against an attacker who manages to subvert your gateway system. If manual keying is in use and an adversary acquires root privilege on your gateway, he reads your keys from `/etc/ipsec.conf` and then reads all messages encrypted with those keys.

If automatic keying is used, an adversary with the same privileges can read `/etc/ipsec.secrets`, but this does not contain any keys, only the secrets used to authenticate key exchanges. Having an adversary able to authenticate your key exchanges need not worry you overmuch. Just having the secrets does not give him any keys. You are still secure against passive attacks. This property of automatic keying is called perfect forward secrecy, abbreviated PFS.

Unfortunately, having the secrets does allow an active attack, specifically a man-in-the-middle attack. Losing these secrets to an attacker may not be quite as disastrous as losing the actual keys, but it is *still a serious security breach*. These secrets should be guarded as carefully as keys.

## Methods not yet implemented

### Unauthenticated key exchange

It would be possible to exchange keys without authenticating the players. This would support opportunistic encryption — allowing any two systems to encrypt their communications without requiring a shared PKI or a previously negotiated secret — and would be secure against passive attacks. It would, however, be highly vulnerable to active man-in-the-middle attacks. RFC 2408 therefore specifies that all ISAKMP key management interactions *must* be authenticated.

There is room for debate here. Should we provide immediate security against passive attacks and encourage widespread use of encryption, at the expense of risking the more difficult active attacks? Or should we wait until we can implement a solution that can both be widespread and offer security against active attacks?

So far, we have chosen the second course, complying with the RFCs and waiting for secure DNS (see below) so that we can do opportunistic encryption right.

### Key exchange using DNS

The IPsec RFCs allow key exchange based on authentication services provided by Secure DNS. Once Secure DNS service becomes widely available, we expect to make this the *primary key management method for Linux FreeS/WAN*. It is the best way we know of to support opportunistic encryption, allowing two systems without a common PKI or previous negotiation to secure their communication.

We currently have code to acquire RSA keys from DNS but do not yet have code to validate Secure DNS signatures.

### Key exchange using a PKI

The IPsec RFCs allow key exchange based on authentication services provided by a PKI or Public Key Infrastructure. With many vendors selling such products and many large organisations building these infrastructures, this will clearly be an important application of IPsec and one Linux FreeS/WAN will eventually support.

On the other hand, this is not as high a priority for Linux FreeS/WAN as solutions based on secure DNS. We do not expect any PKI to become as universal as DNS.

Some patches to handle authentication with X.509 certificates, which most PKIs use, are available.

### **Photuris**

Photuris is another key management protocol, an alternative to IKE and ISAKMP, described in RFCs 2522 and 2523 which are labelled "experimental". Adding Photuris support to Linux FreeS/WAN might be a good project for a volunteer. The likely starting point would be the OpenBSD photurisd code.

### **SKIP**

SKIP is yet another key management protocol, developed by Sun. At one point it was fairly widely used, but it now seems moribund, displaced by IKE. Sun now (as of Solaris 8.0) ship an IPsec implementation using IKE. We have no plans to implement SKIP. If a user were to implement it, we would almost certainly not want to add the code to our distribution.

# Mailing lists and newsgroups

## Mailing lists about FreeS/WAN

### The project mailing lists

The Linux FreeS/WAN project has several email lists for user support, bug reports and software development discussions.

We had a single list on clinet.fi for several years (Thanks, folks!), then one list on freeswan.org, but now we've split into several lists:

#### users

- ◇ The general list for discussing use of the software
- ◇ The place for seeking *help with problems* (but please check the [FAQ](#) first).
- ◇ Anyone can post.

#### bugs

- ◇ For *bug reports*.
- ◇ If you are not certain what is going on — could be a bug, a configuration error, a network problem, ... — please post to the users list instead.
- ◇ Anyone can post.

#### design

- ◇ *Design discussions*, for people working on FreeS/WAN development or others with an interest in design and security issues.
- ◇ It would be a good idea to read the existing design papers (see this [list](#)) before posting.
- ◇ Anyone can post.

#### announce

- ◇ A *low-traffic* list.
- ◇ *Announcements* about FreeS/WAN and related software.
- ◇ All posts here are also sent to the users list. You need not subscribe to both.
- ◇ Only the FreeS/WAN team can post.
- ◇ If you have something you feel should go on this list, send it to [announce-admin@lists.freeswan.org](mailto:announce-admin@lists.freeswan.org). Unless it is obvious, please include a short note explaining why we should post it.

#### briefs

- ◇ A *low-traffic* list.
- ◇ *Weekly summaries* of activity on the users list.
- ◇ All posts here are also sent to the users list. You need not subscribe to both.
- ◇ Only the FreeS/WAN team can post.

To subscribe to any of these, you can:

- just follow the links above
- use our [web interface](#)
- send mail to [listname-request@lists.freeswan.org](mailto:listname-request@lists.freeswan.org) with a one-line message body "subscribe"

Archives of these lists are available via the [web interface](#).

## Which list should I use?

For most questions, please check the [FAQ](#) first, and if that does not have an answer, ask on the users list. "My configuration doesn't work." does not belong on the bugs list, and "Can FreeS/WAN do such-and-such" or "How do I configure it to..." do not belong in design discussions.

Cross-posting the same message to two or more of these lists is discouraged. Quite a few people read more than one list and getting multiple copies is annoying.

## List policies

*US citizens or residents are asked not to post code to the lists, not even one-line bug fixes.* The project cannot accept code which might entangle it in US [export restrictions](#).

Non-subscribers can post to some of these lists. This is necessary; someone working on a gateway install who encounters a problem may not have access to a subscribed account.

Some spam turns up on these lists from time to time. For discussion of why we do not attempt to filter it, see the [FAQ](#). Please do not clutter the lists with complaints about this.

## Archives of the lists

Searchable archives of the old single list have existed for some time. At time of writing, it is not yet clear how they will change for the new multi-list structure.

- [Canada](#)
- [Holland](#)

Note that these use different search engines. Try both.

Archives of the new lists are available via the [web interface](#).

## Indexes of mailing lists

[PAML](#) is the standard reference for *Publicly Accessible Mailing Lists*. When we last checked, it had over 7500 lists on an amazing variety of topics. It also has FAQ information and a search engine.

There is an index of [Linux mailing lists](#) available.

A list of [computer security mailing lists](#), with descriptions.

## Lists for related software and topics

Most links in this section point to subscription addresses for the various lists. Send the one-line message "subscribe *list\_name*" to subscribe to any of them.

## Products that include FreeS/WAN

Our introduction document gives a [list of products that include FreeS/WAN](#). If you have, or are considering, one of those, check the supplier's web site for information on mailing lists for their users.

## Linux mailing lists

- [linux-admin@vger.kernel.org](mailto:linux-admin@vger.kernel.org), for Linux system administrators
- [netfilter@lists.samba.org](mailto:netfilter@lists.samba.org), about Netfilter, which replaces IPchains in kernels 2.3.15 and later
- [security-audit@ferret.lmh.ox.ac.uk](mailto:security-audit@ferret.lmh.ox.ac.uk), for people working on security audits of various Linux programs
- [securedistros@humbolt.geo.uu.nl](mailto:securedistros@humbolt.geo.uu.nl), for discussion of issues common to all the half dozen projects working on secure Linux distributions.

Each of the secure distribution projects also has its own web site and mailing list. Some of the sites are:

- [Bastille Linux](#) scripts to harden Redhat, e.g. by changing permissions and modifying initialisation scripts
- [Immunix](#) take a different approach, using a modified compiler to build kernel and utilities with better resistance to various types of overflow and exploit
- the [NSA](#) have contractors working on a [Security Enhanced Linux](#), primarily adding stronger access control mechanisms. You can download the current version (which interestingly is under GPL and not export restricted) or subscribe to the mailing list from the [project web page](#).

## Lists for IETF working groups

Each [IETF](#) working group has an associated mailing list where much of the work takes place.

- [ipsec@lists.tislabs.com](mailto:ipsec@lists.tislabs.com), the IPsec [working group](#). This is where the protocols are discussed, new drafts announced, and so on. By now, the IPsec working group is winding down since the work is essentially complete. A [list archive](#) is available.
- [IPsec policy](#) list, and its [archive](#)
- [IP secure remote access](#) list, and its [archive](#)

## Other mailing lists

- [ipc-announce@privacy.org](mailto:ipc-announce@privacy.org) a low-traffic list with announcements of developments in privacy, encryption and online civil rights
- a VPN mailing list's [home page](#)

## Usenet newsgroups

- sci.crypt
- sci.crypt.research
- comp.dcom.vpn
- talk.politics.crypto

# Web links

## The Linux FreeS/WAN Project

The main project web site is [www.freeswan.org](http://www.freeswan.org).

Links to other project-related [sites](#) are provided in our introduction section.

### Add-ons and patches for FreeS/WAN

Some user-contributed patches have been integrated into the FreeS/WAN distribution. For a variety of reasons, those listed below have not.

Note that not all patches are a good idea.

- There are a number of "features" of IPsec which we do not implement because they reduce security. See this [discussion](#). We do not recommend using patches that implement these. One example is aggressive mode.
- We do not recommend adding "features" of any sort unless they are clearly necessary, or at least have clear benefits. For example, FreeS/WAN would not become more secure if it offered a choice of 14 ciphers. If even one was flawed, it would certainly become less secure for anyone using that cipher. Even with 14 wonderful ciphers, it would be harder to maintain and administer, hence more vulnerable to various human errors.

This is not to say that patches are necessarily bad, only that using them requires some deliberation. For example, there might be perfectly good reasons to add a specific cipher in your application: perhaps GOST to comply with government standards in Eastern Europe, or AES for performance benefits.

### Current patches

Patches believed current::

- patches for [X.509 certificate support](#), also available from a [mirror site](#)
- patches to add [AES and other ciphers](#). There is preliminary data indicating AES gives a substantial [performance gain](#).

There is also one add-on that takes the form of a modified FreeS/WAN distribution, rather than just patches to the standard distribution:

- [IPv6 support](#)

Before using any of the above,, check the [mailing lists](#) for news of newer versions and to see whether they have been incorporated into more recent versions of FreeS/WAN.

### Older patches

- [hardware acceleration](#)
- a [series](#) of patches that
  - ♦ provide GOST, a Russian gov't. standard cipher, in MMX assembler

## Introduction to FreeS/WAN

- ◆ add GOST to OpenSSL
- ◆ add GOST to the International kernel patch
- ◆ let FreeS/WAN use International kernel patch ciphers
- Neil Dunbar's patches for certificate support, using code from Open SSL.
- Luc Lanthier's patches for PKIX support.
- patches to add Blowfish, IDEA and CAST-128 to FreeS/WAN
- patches for FreeS/WAN 1.3, Pluto support for external authentication, for example with a smartcard or SKEYID.
- patches and utilities for using FreeS/WAN with PGPnet
- Blowfish encryption and Tiger hash
- patches for aggressive mode support

These patches are for older versions of FreeS/WAN and will likely not work with the current version. Older versions of FreeS/WAN may be available on some of the distribution sites, but we recommend using the current release.

### VPN masquerade patches

Finally, there are some patches to other code that may be useful with FreeS/WAN:

- a patch to make IPsec, PPTP and SSH VPNs work through a Linux firewall with IP masquerade.
- Linux VPN Masquerade HOWTO

Note that this is not required if the same machine does IPsec and masquerading, only if you want to locate your IPsec gateway on a masqueraded network. See our firewalls document for discussion of why this is problematic.

At last report, this patch could not co-exist with FreeS/WAN on the same machine.

## Distributions including FreeS/WAN

The introductory section of our document set lists several Linux distributions which include FreeS/WAN.

### Things FreeS/WAN uses or could use

- /dev/random support page, discussion of and code for the Linux random number driver. Out-of-date when we last checked (January 2000), but still useful.
- other programs related to random numbers:
  - ◆ audio entropy daemon to gather noise from a sound card and feed it into /dev/random
  - ◆ an entropy-gathering daemon
  - ◆ a driver for the random number generator in recent Intel chipsets. This driver is included as standard in 2.4 kernels.
- a Linux L2TP Daemon which might be useful for communicating with Windows 2000 which builds L2TP tunnels over its IPsec connections
- to use opportunistic encryption, you need a recent version of BIND. You can get one from the Internet Software Consortium who maintain BIND.

## Other approaches to VPNs for Linux

- other Linux [IPsec implementations](#)
- [ENskip](#), a free implementation of Sun's [SKIP](#) protocol
- [vpnd](#), a non-IPsec VPN daemon for Linux which creates tunnels using [Blowfish](#) encryption
- [Zebedee](#), a simple GPLd tunnel-building program with Linux and Win32 versions. The name is from [Zlib](#) compression, [Blowfish](#) encryption and [Diffie-Hellman](#) key exchange.
- There are at least two PPTP implementations for Linux
  - ◆ Moreton Bay's [PoPToP](#)
  - ◆ [PPTP-Linux](#)
- [CIPE](#) (crypto IP encapsulation) project, using their own lightweight protocol to encrypt between routers
- [tinc](#), a VPN Daemon

There is a list of [Linux VPN](#) software in the [Linux Security Knowledge Base](#).

## The IPsec Protocols

### General IPsec or VPN information

- The [VPN Consortium](#) is a group for vendors of IPsec products. Among other things, they have a good collection of [IPsec white papers](#).
- A VPN mailing list with a [home page](#), a FAQ, some product comparisons, and many links.
- [VPN pointer page](#)
- a [collection](#) of VPN links, and some explanation

### IPsec overview documents or slide sets

- the FreeS/WAN [document section](#) on these protocols

### IPsec information in languages other than English

- [German](#)
- [Japanese](#)
- Feczak Szabolcs' thesis in [Hungarian](#)
- Davide Cerri's thesis and some presentation slides [Italian](#)

### RFCs and other reference documents

- [Our document](#) listing the RFCs relevant to Linux FreeS/WAN and giving various ways of obtaining both RFCs and Internet Drafts.
- [VPN Standards](#) page maintained by [VPN.C](#). This covers both RFCs and Drafts, and classifies them in a fairly helpful way.
- [RFC archive](#)
- [Internet Drafts](#) related to IPsec
- US government [site](#) with their [FIPS](#) standards
- Archives of the ipsec@tis.com mailing list where discussion of drafts takes place.
  - ◆ [Eastern Canada](#)
  - ◆ [California](#).

## Analysis and critiques of IPsec protocols

- Counterpane's [evaluation](#) of the protocols
- Simpson's [IKE Considered Dangerous](#) paper. Note that this is a link to an archive of our mailing list. There are several replies in addition to the paper itself.
- Fate Labs [Virtual Private Problems: the Broken Dream](#)
- Catherine Meadows' paper *Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer*, in [PDF](#) or [Postscript](#).
- Perlman and Kaufmnan
  - ♦ [Key Exchange in IPsec](#)
  - ♦ a newer [PDF paper](#), *Analysis of the IPsec Key Exchange Standard*.
- Bellovin's [papers](#) page including his:
  - ♦ *Security Problems in the TCP/IP Protocol Suite* (1989)
  - ♦ *Problem Areas for the IP Security Protocols* (1996)
  - ♦ *Probable Plaintext Cryptanalysis of the IP Security Protocols* (1997)
- An [errata list](#) for the IPsec RFCs.

## Background information on IP

- An [IP tutorial](#) that seems to be written mainly for Netware or Microsoft LAN admins entering a new world
- [IANA](#), Internet Assigned Numbers Authority
- [CIDR](#), Classless Inter-Domain Routing
- Also see our [bibliography](#)

## IPsec Implementations

### Linux products

Vendors using FreeS/WAN in turnkey firewall or VPN products are listed in our [introduction](#).

Other vendors have Linux IPsec products which, as far as we know, do not use FreeS/WAN

- [Redcreek](#) provide an open source Linux driver for their PCI hardware VPN card. This card has a 100 Mbit Ethernet port, an Intel 960 CPU plus more specialised crypto chips, and claimed encryption performance of 45 Mbit/sec. The PC sees it as an Ethernet board.
- [Paktronix](#) offer a Linux-based VPN with hardware encryption
- [Watchguard](#) use Linux in their Firebox product.
- [Entrust](#) offer a developers' toolkit for using their [PKI](#) for IPsec authentication
- According to a report on our mailing list, [Axent](#) have a Linux version of their product.

### IPsec in router products

All the major router vendors support IPsec, at least in some models.

- [Cisco](#) IPsec information
- Ascend, now part of [Lucent](#), have some IPsec-based products
- [Bay Networks](#), now part of Nortel, use IPsec in their Contivity switch product line
- [3Com](#) have a number of VPN products, some using IPsec

## IPsec in firewall products

Many firewall vendors offer IPsec, either as a standard part of their product, or an optional extra. A few we know about are:

- [Borderware](#)
- [Ashley Laurent](#)
- [Watchguard](#)
- [Injoy](#) for OS/2

Vendors using FreeS/WAN in turnkey firewall products are listed in our [introduction](#).

## Operating systems with IPsec support

All the major open source operating systems support IPsec. See below for details on [BSD-derived](#) Unix variants.

Among commercial OS vendors, IPsec players include:

- [Microsoft](#) have put IPsec in their Windows 2000 and XP products
- [IBM](#) announce a release of OS390 with IPsec support via a crypto co-processor
- [Sun](#) include IPsec in Solaris 8
- [Hewlett Packard](#) offer IPsec for their Unix machines
- Certicom have IPsec available for the [Palm](#).
- There were reports before the release that Apple's Mac OS X would have IPsec support built in, but it did not seem to be there when we last checked. If you find, it please let us know via the [mailing list](#).

## IPsec on network cards

Network cards with built-in IPsec acceleration are available from at least Intel, 3Com and Redcreek.

## Open source IPsec implementations

### Other Linux IPsec implementations

We like to think of FreeS/WAN as *the* Linux IPsec implementation, but it is not the only one. Others we know of are:

- [pipsecd](#), a lightweight implementation of IPsec for Linux. Does not require kernel recompilation.
- Petr Novak's [ipnsec](#), based on the OpenBSD IPsec code and using [Photuris](#) for key management
- A now defunct project at [U of Arizona](#) (export controlled)
- [NIST Cerebus](#) (export controlled)

### IPsec for BSD Unix

- [KAME](#), several large Japanese companies co-operating on IPv6 and IPsec
- [US Naval Research Lab](#) implementation of IPv6 and of IPsec for IPv4 (export controlled)
- [OpenBSD](#) includes IPsec as a standard part of the distribution
- [IPsec for FreeBSD](#)
- a [FAQ](#) on NetBSD's IPsec implementation

## IPsec for other systems

- [Helsinki U of Technology](#) have implemented IPsec for Solaris, Java and Macintosh

## Interoperability

The IPsec protocols are designed so that different implementations should be able to work together. As they say "the devil is in the details". IPsec has a lot of details, but considerable success has been achieved.

### Interoperability results

Linux FreeS/WAN has been tested for interoperability with many other IPsec implementations. Results to date are in our [interoperability](#) section.

Various other sites have information on interoperability between various IPsec implementations:

- [interop results](#) from a bakeoff in Atlanta, September 1999.
- a French company, HSC's, [interoperability](#) test data covers FreeS/WAN, Open BSD, KAME, Linux pipsec, Checkpoint, Red Creek Ravlin, and Cisco IOS
- [ICSA](#) offer certification programs for various security-related products. See their list of [certified IPsec](#) products. Linux FreeS/WAN is not currently on that list, but several products with which we interoperate are.
- VPNC have a page on why they are not yet doing [interoperability](#) testing and a page on the [spec conformance](#) testing that they are doing
- a [review](#) comparing a dozen commercial IPsec implemetations. Unfortunately, the reviewers did not look at Open Source implementations such as FreeS/WAN or OpenBSD.
- [results](#) from interoperability tests at a conference. FreeS/WAN was not tested there.
- test results from the [IPSEC 2000](#) conference

### Interoperability test sites

- [TAHI](#), a Japanese IPv6 testing project with free IPsec validation software
- [National Institute of Standards and Technology](#)
- [SSH Communications Security](#)

## Linux links

### Basic and tutorial Linux information

- [Linux Getting Started](#) HOWTO document
- A getting started guide from the [U of Oregon](#)
- A large [link collection](#) which includes a lot of introductory and tutorial material on Unix, Linux, the net, . . .

### General Linux sites

- [Freshmeat](#) Linux news
- [Slashdot](#) "News for Nerds"
- [Linux Online](#)

- [Linux HQ](#)
- [tux.org](http://tux.org)

## Documentation

Nearly any Linux documentation you are likely to want can be found at the [Linux Documentation Project](#) or LDP.

- [Meta-FAQ](#) guide to Linux information sources
- The LDP's HowTo documents are a standard Linux reference. See this [list](#). Documents there most relevant to a FreeS/WAN gateway are:
  - ♦ [Kernel HOWTO](#)
  - ♦ [Networking Overview HOWTO](#)
  - ♦ [Security HOWTO](#)
- The LDP do a series of Guides, book-sized publications with more detail (and often more "why do it this way?") than the HowTos. See this [list](#). Documents there most relevant to a FreeS/WAN gateway are:
  - ♦ [System Administrator's Guide](#)
  - ♦ [Network Administrator's Guide](#)
  - ♦ [Linux Administrator's Security Guide](#)

You may not need to go to the LDP to get this material. Most Linux distributions include the HowTos on their CDs and several include the Guides as well. Also, most of the Guides and some collections of HowTos are available in book form from various publishers.

Much of the LDP material is also available in languages other than English. See this [LDP page](#).

## Advanced routing

The Linux IP stack has some new features in 2.4 kernels. Some HowTos have been written:

- several HowTos for the [netfilter](#) firewall code in newer kernels
- [2.4 networking](#) HowTo
- [2.4 routing](#) HowTo

## Security for Linux

See also the [LDP material](#) above.

- [Trinity OS guide to setting up Linux](#)
- [Unix security](#) page
- [PPDD](#) encrypting filesystem
- [Linux Encryption HowTo](#) (outdated when last checked, had an Oct 2000 revision date in March 2002)

## Linux firewalls

Our [FreeS/WAN and firewalls](#) document includes links to several sets of [scripts](#) known to work with FreeS/WAN.

Other information sources:

- [IP Masquerade resource page](#)
- [netfilter](#) firewall code in 2.4 kernels
- Our list of general [firewall references](#) on the web
- [Mason](#), a tool for automatically configuring Linux firewalls
- the web cache software [squid](#) and [squidguard](#) which turns Squid into a filtering web proxy

## Miscellaneous Linux information

- [Linux distribution vendors](#)
- [Linux User Groups](#)

## Crypto and security links

### Crypto and security resources

#### The standard link collections

Two enormous collections of links, each the standard reference in its area:

*Gene Spafford's [COAST hotlist](#)*

Computer and network security.

*Peter Gutmann's [Encryption and Security-related Resources](#)*

Cryptography.

#### Frequently Asked Question (FAQ) documents

- [Cryptography FAQ](#)
- [Firewall FAQ](#)
- [Secure Unix Programming FAQ](#)
- FAQs for specific programs are listed in the [tools](#) section below.

#### Tutorials

- Gary Kessler's [Overview of Cryptography](#)
- Terry Ritter's [introduction](#)
- Peter Gutman's [cryptography](#) tutorial (500 slides in PDF format)
- Amir Herzberg of IBM's slides for his course [Introduction to Cryptography and Electronic Commerce](#)
- the [concepts section](#) of the [GNU Privacy Guard](#) documentation
- Bruce Schneier's self-study [cryptanalysis](#) course

See also the [interesting papers](#) section below.

#### Crypto and security standards

- [Common Criteria](#), new international computer and network security standards to replace the "Rainbow" series
- [AES Advanced Encryption Standard](#) which will replace DES
- [IEEE P-1363 public key standard](#)
- our collection of links for the [IPsec](#) standards
- history of [formal evaluation](#) of security policies and implementation

## Crypto quotes

There are several collections of cryptographic quotes on the net:

- [the EFF](#)
- [Sam Simpson](#)
- [AM Kutchling](#)

## Cryptography law and policy

### Surveys of crypto law

- International survey of [crypto law](#).
- International survey of [digital signature law](#)

### Organisations opposing crypto restrictions

- The [EFF](#)'s archives on [privacy](#) and [export control](#).
- [Global Internet Liberty Campaign](#)
- [Center for Democracy and Technology](#)
- [Privacy International](#), who give out [Big Brother Awards](#) to snoop organisations

### Other information on crypto policy

- [RFC 1984](#), the [IAB](#) and [IESG](#) Statement on Cryptographic Technology and the Internet.
- John Young's collection of [documents](#) of interest to the cryptography, open government and privacy movements, organized chronologically
- AT&T researcher Matt Blaze's Encryption, Privacy and Security [Resource Page](#)
- A good [overview](#) of the issues from Australia.

See also our documentation section on the [history and politics](#) of cryptography.

## Cryptography technical information

### Collections of crypto links

- [Counterpane](#)
- [Peter Gutman's links](#)
- [PKI links](#)
- [Robert Guerra's links](#)

### Lists of online cryptography papers

- [Counterpane](#)
- [cryptography.com](#)
- [Cryptosoft](#)

## Particularly interesting papers

These papers emphasize important issues around the use of cryptography, and the design and management of secure systems.

- [Key length requirements for security](#)
- [Why Cryptosystems Fail](#)
- [Risks of escrowed encryption](#)
- [Security pitfalls in cryptography](#)
- [Reflections on Trusting Trust](#), Ken Thompson on Trojan horse design
- [Security against Compelled Disclosure](#), how to maintain privacy in the face of legal or other coercion

## Computer and network security

### Security links

- [COAST Hotlist](#)
- DMOZ open directory project [computer security](#) links
- [Bennet Yee](#)
- Mike Fuhr's [link collection](#)
- [links](#) with an emphasis on intrusion detection

### Firewall links

- [COAST firewalls](#)
- [Firewalls Resource page](#)

### VPN links

- [VPN Consortium](#)
- First VPN's [white paper](#) collection

### Security tools

- PGP — mail encryption
  - ◆ [PGP Inc.](#) (part of NAI) for commercial versions
  - ◆ [MIT](#) distributes the NAI product for non-commercial use
  - ◆ [international](#) distribution site
  - ◆ [GNU Privacy Guard \(GPG\)](#)
  - ◆ [PGP FAQ](#)

A message in our mailing list archive has considerable detail on [available versions](#) of PGP and on IPsec support in them.

**Note:** A fairly nasty bug exists in all commercial PGP versions from 5.5 through 6.5.3. If you have one of those, *upgrade now*.

- SSH — secure remote login
  - ◆ [SSH Communications Security](#), for the original software. It is free for trial, academic and non-commercial use.
  - ◆ [Open SSH](#), the Open BSD team's free replacement
  - ◆ [freessh.org](#), links to free implementations for many systems

## Introduction to FreeS/WAN

- ♦ [SSH FAQ](#)
- ♦ [Putty](#), an SSH client for Windows
- Tripwire saves message digests of your system files. Re-calculate the digests and compare to saved values to detect any file changes. There are several versions available:
  - ♦ [commercial version](#)
  - ♦ [Open Source](#)
- [Snort](#) and [LIDS](#) are intrusion detection system for Linux
- [SATAN](#) System Administrators Tool for Analysing Networks
- [NMAP](#) Network Mapper
- [Wietse Venema's page](#) with various tools
- [Internet Traffic Archive](#), various tools to analyze network traffic, mostly scripts to organise and format tcpdump(8) output for specific purposes
- ssmail — sendmail patched to do [opportunistic encryption](#)
  - ♦ [web page](#) with links to code and to a Usenix paper describing it, in PDF
- [Open CA](#) project to develop a freely distributed [Certification Authority](#) for building a open [Public Key Infrastructure](#).

## Links to home pages

David Wagner at Berkeley provides a set of links to [home pages](#) of cryptographers, cypherpunks and computer security people.

# Glossary for the Linux FreeS/WAN project

Entries are in alphabetical order. Some entries are only one line or one paragraph long. Others run to several paragraphs. I have tried to put the essential information in the first paragraph so you can skip the other paragraphs if that seems appropriate.

---

## Jump to a letter in the glossary

**[numeric](#) [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)**

---

## Other glossaries

Other glossaries which overlap this one include:

- The VPN Consortium's glossary of [VPN terms](#).
- glossary portion of the [Cryptography FAQ](#)
- an extensive cryptographic glossary on [Terry Ritter's](#) page.
- The [NSA's glossary of computer security](#) on the [SANS Institute](#) site.
- a small glossary for Internet Security at [PC magazine](#)
- The [glossary](#) from Richard Smith's book [Internet Cryptography](#)

Several Internet glossaries are available as RFCs:

- [Glossary of Networking Terms](#)
- [Internet User's Glossary](#)
- [Internet Security Glossary](#)

More general glossary or dictionary information:

- Free Online Dictionary of Computing (FOLDOC)
  - ♦ [North America](#)
  - ♦ [Europe](#)
  - ♦ [Japan](#)

There are many more mirrors of this dictionary.

- The Jargon File, the definitive resource for hacker slang and folklore
  - ♦ [North America](#)
  - ♦ [Holland](#)
  - ♦ [home page](#)

There are also many mirrors of this. See the home page for a list.

- A general [technology glossary](#)
  - An [online dictionary resource page](#) with pointers to many dictionaries for many languages
  - A [search engine](#) that accesses several hundred online dictionaries
  - O'Reilly [Dictionary of PC Hardware and Data Communications Terms](#)
  - [Connected](#) Internet encyclopedia
  - [whatis.com](#)
-

## Definitions

0

### 3DES (Triple DES)

Using three DES encryptions on a single data block, with at least two different keys, to get higher security than is available from a single DES pass. The three-key version of 3DES is the default encryption algorithm for Linux FreeS/WAN.

IPsec always does 3DES with three different keys, as required by RFC 2451. For an explanation of the two-key variant, see two key triple DES. Both use an EDE encrypt-decrypt-encrypt sequence of operations.

Single DES is insecure.

Double DES is ineffective. Using two 56-bit keys, one might expect an attacker to have to do  $2^{112}$  work to break it. In fact, only  $2^{57}$  work is required with a meet-in-the-middle attack, though a large amount of memory is also required. Triple DES is vulnerable to a similar attack, but that just reduces the work factor from the  $2^{168}$  one might expect to  $2^{112}$ . That provides adequate protection against brute force attacks, and no better attack is known.

3DES can be somewhat slow compared to other ciphers. It requires three DES encryptions per block. DES was designed for hardware implementation and includes some operations which are difficult in software. However, the speed we get is quite acceptable for many uses. See our performance document for details.

A

### Active attack

An attack in which the attacker does not merely eavesdrop (see passive attack) but takes action to change, delete, reroute, add, forge or divert data. Perhaps the best-known active attack is man-in-the-middle. In general, authentication is a useful defense against active attacks.

AES

The **A**dvanced **E**ncryption **S**tandard — a new block cipher standard to replace DES — developed by NIST, the US National Institute of Standards and Technology. DES used 64-bit blocks and a 56-bit key. AES ciphers use a 128-bit block and 128, 192 or 256-bit keys. The larger block size helps resist birthday attacks while the large key size prevents brute force attacks.

Fifteen proposals meeting NIST's basic criteria were submitted in 1998 and subjected to intense discussion and analysis, "round one" evaluation. In August 1999, NIST narrowed the field to five "round two" candidates:

- ◊ Mars from IBM
- ◊ RC6 from RSA
- ◊ Rijndael from two Belgian researchers
- ◊ Serpent, a British-Norwegian-Israeli collaboration
- ◊ Twofish from the consulting firm Counterpane

Three of the five finalists — Rijndael, Serpent and Twofish — have completely open licenses.

In October 2000, NIST announced the winner — Rijndael.

For more information, see:

## Introduction to FreeS/WAN

- ◇ NIST's [AES home page](#)
- ◇ the Block Cipher Lounge [AES page](#)
- ◇ Brian Gladman's [code and benchmarks](#)
- ◇ Helger Lipmaa's [survey of implementations](#)

AES will be added to a future release of [Linux FreeS/WAN](#). Likely we will add all three of the finalists with good licenses. User-written [AES patches](#) are already available.

Adding AES may also require adding stronger hashes, [SHA-256, SHA-384 and SHA-512](#).

### *AH*

The **IPsec Authentication Header**, added after the IP header. For details, see our [IPsec](#) document and/or RFC 2402.

### *Alice and Bob*

A and B, the standard example users in writing on cryptography and coding theory. Carol and Dave join them for protocols which require more players.

Bruce Schneier extends these with many others such as Eve the Eavesdropper and Victor the Verifier. His extensions seem to be in the process of becoming standard as well. See page 23 of [Applied Cryptography](#)

Alice and Bob have an amusing [biography](#) on the web.

### *ARPA*

see [DARPA](#)

### *ASIO*

Australian Security Intelligence Organisation.

### *Asymmetric cryptography*

See [public key cryptography](#).

### *Authentication*

Ensuring that a message originated from the expected sender and has not been altered on route. [IPsec](#) uses authentication in two places:

- ◇ peer authentication, authenticating the players in [IKE's Diffie-Hellman](#) key exchanges to prevent [man-in-the-middle attacks](#). This can be done in a number of ways. The methods supported by FreeS/WAN are discussed in our [advanced configuration](#) document.
- ◇ packet authentication, authenticating packets on an established [SA](#), either with a separate [authentication header](#) or with the optional authentication in the [ESP](#) protocol. In either case, packet authentication uses a [hashed message authentication code](#) technique.

Outside IPsec, passwords are perhaps the most common authentication mechanism. Their function is essentially to authenticate the person's identity to the system. Passwords are generally only as secure as the network they travel over. If you send a cleartext password over a tapped phone line or over a network with a packet sniffer on it, the security provided by that password becomes zero. Sending an encrypted password is no better; the attacker merely records it and reuses it at his convenience. This is called a [replay](#) attack.

A common solution to this problem is a [challenge-response](#) system. This defeats simple eavesdropping and replay attacks. Of course an attacker might still try to break the cryptographic algorithm used, or the [random number](#) generator.

### *Automatic keying*

A mode in which keys are automatically generated at connection establishment and new keys automatically created periodically thereafter. Contrast with [manual keying](#) in which a single stored key is used.

## Introduction to FreeS/WAN

IPsec uses the Diffie–Hellman key exchange protocol to create keys. An authentication mechanism is required for this. FreeS/WAN normally uses RSA for this. Other methods supported are discussed in our advanced configuration document.

Having an attacker break the authentication is emphatically not a good idea. An attacker that breaks authentication, and manages to subvert some other network entities (DNS, routers or gateways), can use a man-in-the middle attack to break the security of your IPsec connections.

However, having an attacker break the authentication in automatic keying is not quite as bad as losing the key in manual keying.

- ◇ An attacker who reads `/etc/ipsec.conf` and gets the keys for a manually keyed connection can, without further effort, read all messages encrypted with those keys, including any old messages he may have archived.
- ◇ Automatic keying has a property called perfect forward secrecy. An attacker who breaks the authentication gets none of the automatically generated keys and cannot immediately read any messages. He has to mount a successful man-in-the-middle attack in real time before he can read anything. He cannot read old archived messages at all and will not be able to read any future messages not caught by man-in-the-middle tricks.

That said, the secrets used for authentication, stored in ipsec.secrets(5), should still be protected as tightly as cryptographic keys.

### *B*

#### Bay Networks

A vendor of routers, hubs and related products, now a subsidiary of Nortel. Interoperation between their IPsec products and Linux FreeS/WAN was problematic at last report; see our interoperation section.

#### *benchmarks*

Our default block cipher, triple DES, is slower than many alternate ciphers that might be used. Speeds achieved, however, seem adequate for many purposes. For example, the assembler code from the LIBDES library we use encrypts 1.6 megabytes per second on a Pentium 200, according to the test program supplied with the library.

For more detail, see our document on FreeS/WAN performance.

### *BIND*

**Berkeley Internet Name Daemon**, a widely used implementation of DNS (Domain Name Service). See our bibliography for a useful reference. See the BIND home page for more information and the latest version.

#### *Birthday attack*

A cryptographic attack based on the mathematics exemplified by the birthday paradox. This math turns up whenever the question of two cryptographic operations producing the same result becomes an issue:

- ◇ collisions in message digest functions.
- ◇ identical output blocks from a block cipher
- ◇ repetition of a challenge in a challenge–response system

Resisting such attacks is part of the motivation for:

- ◇ hash algorithms such as SHA and RIPEMD–160 giving a 160-bit result rather than the 128 bits of MD4, MD5 and RIPEMD–128.
- ◇ AES block ciphers using a 128-bit block instead of the 64-bit block of most current ciphers
- ◇ IPsec using a 32-bit counter for packets sent on an automatically keyed SA and requiring that the connection always be rekeyed before the counter overflows.

### *Birthday paradox*

Not really a paradox, just a rather counter-intuitive mathematical fact. In a group of 23 people, the chance of a least one pair having the same birthday is over 50%.

The second person has 1 chance in 365 (ignoring leap years) of matching the first. If they don't match, the third person's chances of matching one of them are 2/365. The 4th, 3/365, and so on. The total of these chances grows more quickly than one might guess.

### *Block cipher*

A symmetric cipher which operates on fixed-size blocks of plaintext, giving a block of ciphertext for each. Contrast with stream cipher. Block ciphers can be used in various modes when multiple block are to be encrypted.

DES is among the the best known and widely used block ciphers, but is now obsolete. Its 56-bit key size makes it highly insecure today. Triple DES is the default block cipher for Linux FreeS/WAN.

The current generation of block ciphers — such as Blowfish, CAST-128 and IDEA — all use 64-bit blocks and 128-bit keys. The next generation, AES, uses 128-bit blocks and supports key sizes up to 256 bits.

The Block Cipher Lounge web site has more information.

### *Blowfish*

A block cipher using 64-bit blocks and keys of up to 448 bits, designed by Bruce Schneier and used in several products.

This is not required by the IPsec RFCs and not currently used in Linux FreeS/WAN.

### *Brute force attack (exhaustive search)*

Breaking a cipher by trying all possible keys. This is always possible in theory (except against a one-time pad), but it becomes practical only if the key size is inadequate. For an important example, see our document on the insecurity of DES with its 56-bit key. For an analysis of key sizes required to resist plausible brute force attacks, see this paper.

Longer keys protect against brute force attacks. Each extra bit in the key doubles the number of possible keys and therefore doubles the work a brute force attack must do. A large enough key defeats *any* brute force attack.

For example, the EFF's DES Cracker searches a 56-bit key space in an average of a few days. Let us assume an attacker that can find a 64-bit key (256 times harder) by brute force search in a second (a few hundred thousand times faster). For a 96-bit key, that attacker needs  $2^{32}$  seconds, about 135 years. Against a 128-bit key, he needs  $2^{32}$  times that, over 500,000,000,000 years. Your data is then obviously secure against brute force attacks. Even if our estimate of the attacker's speed is off by a factor of a million, it still takes him over 500,000 years to crack a message.

This is why

- ◇ single DES is now considered dangerously insecure
- ◇ all of the current generation of block ciphers use a 128-bit or longer key
- ◇ AES ciphers support key sizes 128, 192 and 256 bits
- ◇ any cipher we add to Linux FreeS/WAN will have *at least* a 128-bit key

### ***Cautions:***

*Inadequate keylength always indicates a weak cipher* but it is important to note that *adequate keylength does not necessarily indicate a strong cipher*. There are many attacks other than brute force,

## Introduction to FreeS/WAN

and adequate keylength *only* guarantees resistance to brute force. Any cipher, whatever its key size, will be weak if design or implementation flaws allow other attacks.

Also, *once you have adequate keylength* (somewhere around 90 or 100 bits), *adding more key bits make no practical difference*, even against brute force. Consider our 128-bit example above that takes 500,000,000,000 years to break by brute force. We really don't care how many zeroes there are on the end of that, as long as the number remains ridiculously large. That is, we don't care exactly how large the key is as long as it is large enough.

There may be reasons of convenience in the design of the cipher to support larger keys. For example Blowfish allows up to 448 bits and RC4 up to 2048, but beyond 100-odd bits it makes no difference to practical security.

*Bureau of Export Administration*

see BXA

*BXA*

The US Commerce Department's **Bureau of Export Administration** which administers the EAR Export Administration Regulations controlling the export of, among other things, cryptography.

*C*

*CA*

**Certification Authority**, an entity in a public key infrastructure that can certify keys by signing them. Usually CAs form a hierarchy. The top of this hierarchy is called the root CA.

See Web of Trust for an alternate model.

*CAST-128*

A block cipher using 64-bit blocks and 128-bit keys, described in RFC 2144 and used in products such as Entrust and recent versions of PGP.

This is not required by the IPsec RFCs and not currently used in Linux FreeS/WAN.

*CAST-256*

Entrust's candidate cipher for the AES standard, largely based on the CAST-128 design.

*CBC mode*

**Cipher Block Chaining mode**, a method of using a block cipher in which for each block except the first, the result of the previous encryption is XORed into the new block before it is encrypted. CBC is the mode used in IPsec.

An initialisation vector (IV) must be provided. It is XORed into the first block before encryption. The IV need not be secret but should be different for each message and unpredictable.

*CIDR*

**Classless Inter-Domain Routing**, an addressing scheme used to describe networks not restricted to the old Class A, B, and C sizes. A CIDR block is written *address/mask*, where *address* is a 32-bit Internet address. The first *mask* bits of *address* are part of the gateway address, while the remaining bits designate other host addresses. For example, the CIDR block 192.0.2.96/27 describes a network with gateway 192.0.2.96, hosts 192.0.2.96 through 192.0.2.126 and broadcast 192.0.2.127.

FreeS/WAN policy group files accept CIDR blocks of the format *address/[mask]*, where *address* may take the form *name.domain.tld*. An absent *mask* is assumed to be /32.

*Certification Authority*

see CA

*Challenge-response authentication*

An authentication system in which one player generates a random number, encrypts it and sends the result as a challenge. The other player decrypts and sends back the result. If the result is correct, that

## Introduction to FreeS/WAN

proves to the first player that the second player knew the appropriate secret, required for the decryption. Variations on this technique exist using public key or symmetric cryptography. Some provide two-way authentication, assuring each player of the other's identity.

This is more secure than passwords against two simple attacks:

- ◊ If cleartext passwords are sent across the wire (e.g. for telnet), an eavesdropper can grab them. The attacker may even be able to break into other systems if the user has chosen the same password for them.
- ◊ If an encrypted password is sent, an attacker can record the encrypted form and use it later. This is called a replay attack.

A challenge-response system never sends a password, either cleartext or encrypted. An attacker cannot record the response to one challenge and use it as a response to a later challenge. The random number is different each time.

Of course an attacker might still try to break the cryptographic algorithm used, or the random number generator.

### *Cipher Modes*

Different ways of using a block cipher when encrypting multiple blocks.

Four standard modes were defined for DES in FIPS 81. They can actually be applied with any block cipher.

<u>ECB</u>	Electronic CodeBook	encrypt each block independently
<u>CBC</u>	Cipher Block Chaining	XOR previous block ciphertext into new block plaintext before encrypting new block
CFB	Cipher FeedBack	
OFB	Output FeedBack	

IPsec uses CBC mode since this is only marginally slower than ECB and is more secure. In ECB mode the same plaintext always encrypts to the same ciphertext, unless the key is changed. In CBC mode, this does not occur.

Various other modes are also possible, but none of them are used in IPsec.

### *Ciphertext*

The encrypted output of a cipher, as opposed to the unencrypted plaintext input.

### *Cisco*

A vendor of routers, hubs and related products. Their IPsec products interoperate with Linux FreeS/WAN; see our interop section.

### *Client*

This term has at least two distinct uses in discussing IPsec:

- ◊ The ***clients of an IPsec gateway*** are the machines it protects, typically on one or more subnets behind the gateway. In this usage, all the machines on an office network are clients of that office's IPsec gateway. Laptop or home machines connecting to the office, however, are *not* clients of that gateway. They are remote gateways, running the other end of an IPsec connection. Each of them is also its own client.
- ◊ ***IPsec client software*** is used to describe software which runs on various standalone machines to let them connect to IPsec networks. In this usage, a laptop or home machine connecting to the office is a client, and the office gateway is the server.

## Introduction to FreeS/WAN

We generally use the term in the first sense. Vendors of Windows IPsec solutions often use it in the second. See this [discussion](#).

### *Common Criteria*

A set of international security classifications which are replacing the old US [Rainbow Book](#) standards and similar standards in other countries.

Web references include this [US government site](#) and this [global home page](#).

### *Conventional cryptography*

See [symmetric cryptography](#)

### *Collision resistance*

The property of a [message digest](#) algorithm which makes it hard for an attacker to find or construct two inputs which hash to the same output.

### *Copyleft*

see GNU [General Public License](#)

### *CSE*

[Communications Security Establishment](#), the Canadian organisation for [signals intelligence](#).

### *D*

#### *DARPA (sometimes just ARPA)*

The US government's **Defense Advanced Research Projects Agency**. Projects they have funded over the years have included the Arpanet which evolved into the Internet, the TCP/IP protocol suite (as a replacement for the original Arpanet suite), the Berkeley 4.x BSD Unix projects, and [Secure DNS](#).

For current information, see their [web site](#).

### *Denial of service (DoS) attack*

An attack that aims at denying some service to legitimate users of a system, rather than providing a service to the attacker.

- ◊ One variant is a flooding attack, overwhelming the system with too many packets, too much email, or whatever.
- ◊ A closely related variant is a resource exhaustion attack. For example, consider a "TCP SYN flood" attack. Setting up a TCP connection involves a three–packet exchange:
  - Initiator: Connection please (SYN)
  - Responder: OK (ACK)
  - Initiator: OK here too

If the attacker puts bogus source information in the first packet, such that the second is never delivered, the responder may wait a long time for the third to come back. If responder has already allocated memory for the connection data structures, and if many of these bogus packets arrive, the responder may run out of memory.

- ◊ Another variant is to feed the system undigestible data, hoping to make it sick. For example, IP packets are limited in size to 64K bytes and a fragment carries information on where it starts within that 64K and how long it is. The "ping of death" delivers fragments that say, for example, that they start at 60K and are 20K long. Attempting to re–assemble these without checking for overflow can be fatal.

The two example attacks discussed were both quite effective when first discovered, capable of crashing or disabling many operating systems. They were also well–publicised, and today far fewer systems are vulnerable to them.

### *DES*

The **Data Encryption Standard**, a [block cipher](#) with 64–bit blocks and a 56–bit key. Probably the most widely used [symmetric cipher](#) ever devised. DES has been a US government standard for their own use (only for unclassified data), and for some regulated industries such as banking, since the late 70's. It is now being replaced by [AES](#).

DES is seriously insecure against current attacks.

Linux FreeS/WAN does not include DES, even though the RFCs specify it. **We strongly recommend that single DES not be used.**

See also 3DES and DESX, stronger ciphers based on DES.

### *DESX*

An improved DES suggested by Ron Rivest of RSA Data Security. It XORs extra key material into the text before and after applying the DES cipher.

This is not required by the IPsec RFCs and not currently used in Linux FreeS/WAN. DESX would be the easiest additional transform to add; there would be very little code to write. It would be much faster than 3DES and almost certainly more secure than DES. However, since it is not in the RFCs other IPsec implementations cannot be expected to have it.

### *DH*

see Diffie–Hellman

### *DHCP*

**D**ynamic **H**ost **C**onfiguration **P**rotocol, a method of assigning dynamic IP addresses, and providing additional information such as addresses of DNS servers and of gateways. See this DHCP resource page.

### *Diffie–Hellman (DH) key exchange protocol*

A protocol that allows two parties without any initial shared secret to create one in a manner immune to eavesdropping. Once they have done this, they can communicate privately by using that shared secret as a key for a block cipher or as the basis for key exchange.

The protocol is secure against all passive attacks, but it is not at all resistant to active man-in-the-middle attacks. If a third party can impersonate Bob to Alice and vice versa, then no useful secret can be created. Authentication of the participants is a prerequisite for safe Diffie–Hellman key exchange. IPsec can use any of several authentication mechanisms. Those supported by FreeS/WAN are discussed in our configuration section.

The Diffie–Hellman key exchange is based on the discrete logarithm problem and is secure unless someone finds an efficient solution to that problem.

Given a prime  $p$  and generator  $g$  (explained under discrete log below), Alice:

- ◇ generates a random number  $a$
- ◇ calculates  $A = g^a \text{ modulo } p$
- ◇ sends  $A$  to Bob

Meanwhile Bob:

- ◇ generates a random number  $b$
- ◇ calculates  $B = g^b \text{ modulo } p$
- ◇ sends  $B$  to Alice

Now Alice and Bob can both calculate the shared secret  $s = g^{(ab)}$ . Alice knows  $a$  and  $B$ , so she calculates  $s = B^a$ . Bob knows  $A$  and  $b$  so he calculates  $s = A^b$ .

An eavesdropper will know  $p$  and  $g$  since these are made public, and can intercept  $A$  and  $B$  but, short of solving the discrete log problem, these do not let him or her discover the secret  $s$ .

### *Digital signature*

Sender:

## Introduction to FreeS/WAN

- ◇ calculates a message digest of a document
- ◇ encrypts the digest with his or her private key, using some public key cryptosystem.
- ◇ attaches the encrypted digest to the document as a signature

Receiver:

- ◇ calculates a digest of the document (not including the signature)
- ◇ decrypts the signature with the signer's public key
- ◇ verifies that the two results are identical

If the public-key system is secure and the verification succeeds, then the receiver knows

- ◇ that the document was not altered between signing and verification
- ◇ that the signer had access to the private key

Such an encrypted message digest can be treated as a signature since it cannot be created without *both* the document *and* the private key which only the sender should possess. The legal issues are complex, but several countries are moving in the direction of legal recognition for digital signatures.

### *discrete logarithm problem*

The problem of finding logarithms in a finite field. Given a field definition (such definitions always include some operation analogous to multiplication) and two numbers, a base and a target, find the power which the base must be raised to in order to yield the target.

The discrete log problem is the basis of several cryptographic systems, including the Diffie-Hellman key exchange used in the IKE protocol. The useful property is that exponentiation is relatively easy but the inverse operation, finding the logarithm, is hard. The cryptosystems are designed so that the user does only easy operations (exponentiation in the field) but an attacker must solve the hard problem (discrete log) to crack the system.

There are several variants of the problem for different types of field. The IKE/Oakley key determination protocol uses two variants, either over a field modulo a prime or over a field defined by an elliptic curve. We give an example modulo a prime below. For the elliptic curve version, consult an advanced text such as Handbook of Applied Cryptography.

Given a prime  $p$ , a generator  $g$  for the field modulo that prime, and a number  $x$  in the field, the problem is to find  $y$  such that  $g^y = x$ .

For example, let  $p = 13$ . The field is then the integers from 0 to 12. Any integer equals one of these modulo 13. That is, the remainder when any integer is divided by 13 must be one of these.

2 is a generator for this field. That is, the powers of two modulo 13 run through all the non-zero numbers in the field. Modulo 13 we have:

y	x
$2^0$	== 1
$2^1$	== 2
$2^2$	== 4
$2^3$	== 8
$2^4$	== 3 that is, the remainder from 16/13 is 3
$2^5$	== 6 the remainder from 32/13 is 6
$2^6$	== 12 and so on
$2^7$	== 11
$2^8$	== 9
$2^9$	== 5
$2^{10}$	== 10
$2^{11}$	== 7

$$2^{12} \equiv 1$$

Exponentiation in such a field is not difficult. Given, say,  $y = 11$ , calculating  $x = 7$  is straightforward. One method is just to calculate  $2^{11} = 2048$ , then  $2048 \bmod 13 \equiv 7$ . When the field is modulo a large prime (say a few 100 digits) you need a slightly cleverer method and even that is moderately expensive in computer time, but the calculation is still not problematic in any basic way.

The discrete log problem is the reverse. In our example, given  $x = 7$ , find the logarithm  $y = 11$ . When the field is modulo a large prime (or is based on a suitable elliptic curve), this is indeed problematic. No solution method that is not catastrophically expensive is known. Quite a few mathematicians have tackled this problem. No efficient method has been found and mathematicians do not expect that one will be. It seems likely no efficient solution to either of the main variants the discrete log problem exists.

Note, however, that no-one has proven such methods do not exist. If a solution to either variant were found, the security of any crypto system using that variant would be destroyed. This is one reason IKE supports two variants. If one is broken, we can switch to the other.

*discretionary access control*

access control mechanisms controlled by the user, for example Unix rwx file permissions. These contrast with mandatory access controls.

*DNS*

**Domain Name Service**, a distributed database through which names are associated with numeric addresses and other information in the Internet Protocol Suite. See also the DNS background section of our documentation.

*DOS attack*

see Denial Of Service attack

*dynamic IP address*

an IP address which is automatically assigned, either by DHCP or by some protocol such as PPP or PPPoE which the machine uses to connect to the Internet. This is the opposite of a static IP address, pre-set on the machine itself.

*E*

*EAR*

The US government's **Export Administration Regulations**, administered by the Bureau of Export Administration. These have replaced the earlier ITAR regulations as the controls on export of cryptography.

*ECB mode*

**Electronic CodeBook** mode, the simplest way to use a block cipher. See Cipher Modes.

*EDE*

The sequence of operations normally used in either the three-key variant of triple DES used in IPsec or the two-key variant used in some other systems.

The sequence is:

- ◊ **E**ncrypt with key1
- ◊ **D**ecrypt with key2
- ◊ **E**ncrypt with key3

For the two-key version,  $\text{key1} = \text{key3}$ .

The "advantage" of this EDE order of operations is that it makes it simple to interoperate with older devices offering only single DES. Set  $\text{key1} = \text{key2} = \text{key3}$  and you have the worst of both worlds, the overhead of triple DES with the "security" of single DES. Since both the security of single DES and

## Introduction to FreeS/WAN

the overheads of triple DES are seriously inferior to many other ciphers, this is a spectacularly dubious "advantage".

### *Entrust*

A Canadian company offering enterprise PKI products using CAST-128 symmetric crypto, RSA public key and X.509 directories. Web site

### *EFF*

Electronic Frontier Foundation, an advocacy group for civil rights in cyberspace.

### *Encryption*

Techniques for converting a readable message (plaintext) into apparently random material (ciphertext) which cannot be read if intercepted. A key is required to read the message.

Major variants include symmetric encryption in which sender and receiver use the same secret key and public key methods in which the sender uses one of a matched pair of keys and the receiver uses the other. Many current systems, including IPsec, are hybrids combining the two techniques.

### *ESP*

Encapsulated Security Payload, the IPsec protocol which provides encryption. It can also provide authentication service and may be used with null encryption (which we do not recommend). For details see our IPsec document and/or RFC 2406.

### *Extruded subnet*

A situation in which something IP sees as one network is actually in two or more places.

For example, the Internet may route all traffic for a particular company to that firm's corporate gateway. It then becomes the company's problem to get packets to various machines on their subnets in various departments. They may decide to treat a branch office like a subnet, giving it IP addresses "on" their corporate net. This becomes an extruded subnet.

Packets bound for it are delivered to the corporate gateway, since as far as the outside world is concerned, that subnet is part of the corporate network. However, instead of going onto the corporate LAN (as they would for, say, the accounting department) they are then encapsulated and sent back onto the Internet for delivery to the branch office.

For information on doing this with Linux FreeS/WAN, look in our advanced configuration section.

### *Exhaustive search*

See brute force attack.

### *F*

### *FIPS*

**Federal Information Processing Standard**, the US government's standards for products it buys. These are issued by NIST. Among other things, DES and SHA are defined in FIPS documents. NIST have a FIPS home page.

### *Free Software Foundation (FSF)*

An organisation to promote free software, free in the sense of these quotes from their web pages

"Free software" is a matter of liberty, not price. To understand the concept, you should think of "free speech", not "free beer."

"Free software" refers to the users' freedom to run, copy, distribute, study, change and improve the software.

See also GNU, GNU General Public License, and the FSF site.

### *FreeS/WAN*

see Linux FreeS/WAN

### *Fullnet*

## Introduction to FreeS/WAN

The CIDR block containing all IPs of its IP version. The IPv4 fullnet is written 0.0.0.0/0. Also known as "all" and "default", fullnet may be used in a routing table to specify a default route, and in a FreeS/WAN policy group file to specify a default IPsec policy.

*FSF*

see Free software Foundation

*G*

*GCHQ*

Government Communications Headquarters, the British organisation for signals intelligence.

*generator of a prime field*

see discrete logarithm problem

*GILC*

Global Internet Liberty Campaign, an international organisation advocating, among other things, free availability of cryptography. They have a campaign to remove cryptographic software from the Wassenaar Arrangement.

*Global Internet Liberty Campaign*

see GILC.

*Global Trust Register*

An attempt to create something like a root CA for PGP by publishing both as a book and on the web the fingerprints of a set of verified keys for well-known users and organisations.

*GMP*

The GNU Multi-Precision library code, used in Linux FreeS/WAN by Pluto for public key calculations. See the GMP home page.

*GNU*

GNU's Not Unix, the Free Software Foundation's project aimed at creating a free system with at least the capabilities of Unix. Linux uses GNU utilities extensively.

*GOST*

a Soviet government standard block cipher. Applied Cryptography has details.

*GPG*

see GNU Privacy Guard

*GNU General Public License(GPL, copyleft)*

The license developed by the Free Software Foundation under which Linux, Linux FreeS/WAN and many other pieces of software are distributed. The license allows anyone to redistribute and modify the code, but forbids anyone from distributing executables without providing access to source code. For more details see the file COPYING included with GPLed source distributions, including ours, or the GNU site's GPL page.

*GNU Privacy Guard*

An open source implementation of Open PGP as defined in RFC 2440. See their web site

*GPL*

see GNU General Public License.

*H*

*Hash*

see message digest

*Hashed Message Authentication Code (HMAC)*

using keyed message digest functions to authenticate a message. This differs from other uses of these functions:

- ◊ In normal usage, the hash function's internal variable are initialised in some standard way. Anyone can reproduce the hash to check that the message has not been altered.
- ◊ For HMAC usage, you initialise the internal variables from the key. Only someone with the key can reproduce the hash. A successful check of the hash indicates not only that the message is unchanged but also that the creator knew the key.

## Introduction to FreeS/WAN

The exact techniques used in IPsec are defined in RFC 2104. They are referred to as HMAC–MD5–96 and HMAC–SHA–96 because they output only 96 bits of the hash. This makes some attacks on the hash functions harder.

### *HMAC*

see Hashed Message Authentication Code

### *HMAC–MD5–96*

see Hashed Message Authentication Code

### *HMAC–SHA–96*

see Hashed Message Authentication Code

### *Hybrid cryptosystem*

A system using both public key and symmetric cipher techniques. This works well. Public key methods provide key management and digital signature facilities which are not readily available using symmetric ciphers. The symmetric cipher, however, can do the bulk of the encryption work much more efficiently than public key methods.

### *I*

### *IAB*

Internet Architecture Board.

### *ICMP*

**I**nternet **C**ontrol **M**essage **P**rotocol. This is used for various IP–connected devices to manage the network.

### *IDEA*

**I**nternational **D**ata **E**ncryption **A**lgorithm, developed in Europe as an alternative to exportable American ciphers such as DES which were too weak for serious use. IDEA is a block cipher using 64–bit blocks and 128–bit keys, and is used in products such as PGP.

IDEA is not required by the IPsec RFCs and not currently used in Linux FreeS/WAN.

IDEA is patented and, with strictly limited exceptions for personal use, using it requires a license from Ascom.

### *IEEE*

Institute of Electrical and Electronic Engineers, a professional association which, among other things, sets some technical standards

### *IESG*

Internet Engineering Steering Group.

### *IETF*

Internet Engineering Task Force, the umbrella organisation whose various working groups make most of the technical decisions for the Internet. The IETF IPsec working group wrote the RFCs we are implementing.

### *IKE*

**I**nternet **K**ey **E**xchange, based on the Diffie–Hellman key exchange protocol. For details, see RFC 2409 and our IPsec document. IKE is implemented in Linux FreeS/WAN by the Pluto daemon.

### *IKE v2*

A proposed replacement for IKE. There are other candidates, such as JFK, and at time of writing (March 2002) the choice between them has not yet been made and does not appear imminent.

### *iOE*

See Initiate–only opportunistic encryption.

### *IP*

**I**nternet **P**rotocol.

### *IP masquerade*

A mostly obsolete term for a method of allowing multiple machines to communicate over the Internet when only one IP address is available for their use. The more current term is Network Address

Translation or NAT.

### *IPng*

"IP the Next Generation", see IPv6.

### *IPv4*

The current version of the Internet protocol suite.

### *IPv6 (IPng)*

Version six of the Internet protocol suite, currently being developed. It will replace the current version four. IPv6 has IPsec as a mandatory component.

See this web site for more details, and our compatibility document for information on FreeS/WAN and the Linux implementation of IPv6.

### *IPsec or IPSEC*

Internet **P**rotocol **S**ECurity, security functions (authentication and encryption) implemented at the IP level of the protocol stack. It is optional for IPv4 and mandatory for IPv6.

This is the standard Linux FreeS/WAN is implementing. For more details, see our IPsec Overview. For the standards, see RFCs listed in our RFCs document.

### *IPX*

Novell's Netware protocol tunnelled over an IP link. Our firewalls document includes an example of using this through an IPsec tunnel.

### *ISAKMP*

Internet Security Association and **K**ey Management Protocol, defined in RFC 2408.

### *ITAR*

International Traffic in Arms Regulations, US regulations administered by the State Department which until recently limited export of, among other things, cryptographic technology and software. ITAR still exists, but the limits on cryptography have now been transferred to the Export Administration Regulations under the Commerce Department's Bureau of Export Administration.

### *IV*

see Initialisation vector

### *Initialisation Vector (IV)*

Some cipher modes, including the CBC mode which IPsec uses, require some extra data at the beginning. This data is called the initialisation vector. It need not be secret, but should be different for each message. Its function is to prevent messages which begin with the same text from encrypting to the same ciphertext. That might give an analyst an opening, so it is best prevented.

### *Initiate-only opportunistic encryption (iOE)*

A form of opportunistic encryption (OE) in which a host proposes opportunistic connections, but lacks the reverse DNS records necessary to support incoming opportunistic connection requests. Common among hosts on cable or pppoe connections where the system administrator does not have write access to the DNS reverse map for the host's external IP.

Configuring for initiate-only opportunistic encryption is described in our quickstart document.

### *J*

### *JFK*

**Just Fast Keying**, a proposed simpler replacement for IKE.

### *K*

### *Kernel*

The basic part of an operating system (e.g. Linux) which controls the hardware and provides services to all other programs.

In the Linux release numbering system, an even second digit as in 2.2.x indicates a stable or production kernel while an odd number as in 2.3.x indicates an experimental or development kernel.

## Introduction to FreeS/WAN

Most users should run a recent kernel version from the production series. The development kernels are primarily for people doing kernel development. Others should consider using development kernels only if they have an urgent need for some feature not yet available in production kernels.

*Keyed message digest*

See [HMAC](#).

*Key length*

see [brute force attack](#)

*KLIPS*

**Kernel IP Security**, the [Linux FreeS/WAN](#) project's changes to the [Linux](#) kernel to support the [IPsec](#) protocols.

*L*

*LDAP*

**Lightweight Directory Access Protocol**, defined in RFCs 1777 and 1778, a method of accessing information stored in directories. LDAP is used by several [PKI](#) implementations, often with X.501 directories and [X.509](#) certificates. It may also be used by [IPsec](#) to obtain key certifications from those PKIs. This is not yet implemented in [Linux FreeS/WAN](#).

*LIBDES*

A publicly available library of [DES](#) code, written by Eric Young, which [Linux FreeS/WAN](#) uses in both [KLIPS](#) and [Pluto](#).

*Linux*

A freely available Unix-like operating system based on a kernel originally written for the Intel 386 architecture by (then) student Linus Torvalds. Once his 32-bit kernel was available, the [GNU](#) utilities made it a usable system and contributions from many others led to explosive growth.

Today Linux is a complete Unix replacement available for several CPU architectures --- Intel, DEC/Compaq Alpha, Power PC, both 32-bit SPARC and the 64-bit UltraSPARC, SrongARM, . . . --- with support for multiple CPUs on some architectures.

[Linux FreeS/WAN](#) is intended to run on all CPUs supported by Linux and is known to work on several. See our [compatibility](#) section for a list.

*Linux FreeS/WAN*

Our implementation of the [IPsec](#) protocols, intended to be freely redistributable source code with a [GNU GPL license](#) and no constraints under US or other [export laws](#). Linux FreeS/WAN is intended to interoperate with other [IPsec](#) implementations. The name is partly taken, with permission, from the [S/WAN](#) multi-vendor IPsec compatability effort. Linux FreeS/WAN has two major components, [KLIPS](#) (Kernel IPsec Support) and the [Pluto](#) daemon which manages the whole thing.

See our [IPsec section](#) for more detail. For the code see our [primary site](#) or one of the mirror sites on [this list](#).

*Linux Security Modules (LSM)*

a project to create an interface in the Linux kernel that supports plug-in modules for various security policies.

This allows multiple security projects to take different approaches to security enhancement without tying the kernel down to one particular approach. As I understand the history, several projects were pressing Linus to incorporate their changes, the various sets of changes were incompatible, and his answer was more-or-less "a plague on all your houses; I'll give you an interface, but I won't incorporate anything".

It seems to be working. There is a fairly active [LSM mailing list](#), and several projects are already using the interface.

*LSM*

see [Linux Security Modules](#)

*M*

*Mailing list*

The [Linux FreeS/WAN](#) project has several public email lists for bug reports and software development discussions. See our document on [mailing lists](#).

*Man-in-the-middle attack*

An [active attack](#) in which the attacker impersonates each of the legitimate players in a protocol to the other.

For example, if [Alice and Bob](#) are negotiating a key via the [Diffie-Hellman](#) key agreement, and are not using [authentication](#) to be certain they are talking to each other, then an attacker able to insert himself in the communication path can deceive both players.

Call the attacker Mallory. For Bob, he pretends to be Alice. For Alice, he pretends to be Bob. Two keys are then negotiated, Alice-to-Mallory and Bob-to-Mallory. Alice and Bob each think the key they have is Alice-to-Bob.

A message from Alice to Bob then goes to Mallory who decrypts it, reads it and/or saves a copy, re-encrypts using the Bob-to-Mallory key and sends it along to Bob. Bob decrypts successfully and sends a reply which Mallory decrypts, reads, re-encrypts and forwards to Alice.

To make this attack effective, Mallory must

- ◇ subvert some part of the network in some way that lets him carry out the deception
  - possible targets: DNS, router, Alice or Bob's machine, mail server, ...
- ◇ beat any authentication mechanism Alice and Bob use
  - strong authentication defeats the attack entirely; this is why [IKE](#) requires authentication
- ◇ work in real time, delivering messages without introducing a delay large enough to alert the victims
  - not hard if Alice and Bob are using email; quite difficult in some situations.

If he manages it, however, it is devastating. He not only gets to read all the messages; he can alter messages, inject his own, forge anything he likes, . . . In fact, he controls the communication completely.

*mandatory access control*

access control mechanisms which are not settable by the user (see [discretionary access control](#)), but are enforced by the system.

For example, a document labelled "secret, zebra" might be readable only by someone with secret clearance working on Project Zebra. Ideally, the system will prevent any transfer outside those boundaries. For example, even if you can read it, you should not be able to e-mail it (unless the recipient is appropriately cleared) or print it (unless certain printers are authorised for that classification).

Mandatory access control is a required feature for some levels of [Rainbow Book](#) or [Common Criteria](#) classification, but has not been widely used outside the military and government. There is a good discussion of the issues in Anderson's [Security Engineering](#).

The [Security Enhanced Linux](#) project is adding mandatory access control to Linux.

*Manual keying*

An IPsec mode in which the keys are provided by the administrator. In FreeS/WAN, they are stored in /etc/ipsec.conf. The alternative, automatic keying, is preferred in most cases. See this discussion.

*MD4*

Message Digest Algorithm Four from Ron Rivest of RSA. MD4 was widely used a few years ago, but is now considered obsolete. It has been replaced by its descendants MD5 and SHA.

*MD5*

Message Digest Algorithm Five from Ron Rivest of RSA, an improved variant of his MD4. Like MD4, it produces a 128-bit hash. For details see RFC 1321.

MD5 is one of two message digest algorithms available in IPsec. The other is SHA. SHA produces a longer hash and is therefore more resistant to birthday attacks, but this is not a concern for IPsec. The HMAC method used in IPsec is secure even if the underlying hash is not particularly strong against this attack.

Hans Dobbertin found a weakness in MD5, and people often ask whether this means MD5 is unsafe for IPsec. It doesn't. The IPsec RFCs discuss Dobbertin's attack and conclude that it does not affect MD5 as used for HMAC in IPsec.

*Meet-in-the-middle attack*

A divide-and-conquer attack which breaks a cipher into two parts, works against each separately, and compares results. Probably the best known example is an attack on double DES. This applies in principle to any pair of block ciphers, e.g. to an encryption system using, say, CAST-128 and Blowfish, but we will describe it for double DES.

Double DES encryption and decryption can be written:

$$\begin{aligned} C &= E(k_2, E(k_1, P)) \\ P &= D(k_1, D(k_2, C)) \end{aligned}$$

Where C is ciphertext, P is plaintext, E is encryption, D is decryption, k1 is one key, and k2 is the other key. If we know a P, C pair, we can try and find the keys with a brute force attack, trying all possible k1, k2 pairs. Since each key is 56 bits, there are  $2^{112}$  such pairs and this attack is painfully inefficient.

The meet-in-the middle attack re-writes the equations to calculate a middle value M:

$$\begin{aligned} M &= E(k_1, P) \\ M &= D(k_2, C) \end{aligned}$$

Now we can try some large number of D(k2,C) decryptions with various values of k2 and store the results in a table. Then start doing E(k1,P) encryptions, checking each result to see if it is in the table.

With enough table space, this breaks double DES with  $2^{56} + 2^{56} = 2^{57}$  work. Against triple DES, you need  $2^{56} + 2^{112} \approx 2^{112}$ .

The memory requirements for such attacks can be prohibitive, but there is a whole body of research literature on methods of reducing them.

*Message Digest Algorithm*

An algorithm which takes a message as input and produces a hash or digest of it, a fixed-length set of bits which depend on the message contents in some highly complex manner. Design criteria include making it extremely difficult for anyone to counterfeit a digest or to change a message without altering its digest. One essential property is collision resistance. The main applications are in message

authentication and digital signature schemes. Widely used algorithms include MD5 and SHA. In IPsec, message digests are used for HMAC authentication of packets.

### *MTU*

**Maximum Transmission Unit**, the largest size of packet that can be sent over a link. This is determined by the underlying network, but must be taken account of at the IP level.

IP packets, which can be up to 64K bytes each, must be packaged into lower-level packets of the appropriate size for the underlying network(s) and re-assembled on the other end. When a packet must pass over multiple networks, each with its own MTU, and many of the MTUs are unknown to the sender, this becomes a fairly complex problem. See path MTU discovery for details.

Often the MTU is a few hundred bytes on serial links and 1500 on Ethernet. There are, however, serial link protocols which use a larger MTU to avoid fragmentation at the ethernet/serial boundary, and newer (especially gigabit) Ethernet networks sometimes support much larger packets because these are more efficient in some applications.

### *N*

#### *NAI*

Network Associates, a conglomerate formed from PGP Inc., TIS (Trusted Information Systems, a firewall vendor) and McAfee anti-virus products. Among other things, they offer an IPsec-based VPN product.

#### *NAT*

**Network Address Translation**, a process by which firewall machines may change the addresses on packets as they go through. For discussion, see our background section.

#### *NIST*

The US National Institute of Standards and Technology, responsible for FIPS standards including DES and its replacement, AES.

#### *Nonce*

A random value used in an authentication protocol.

#### *Non-routable IP address*

An IP address not normally allowed in the "to" or "from" IP address field header of IP packets.

Almost invariably, the phrase "non-routable address" means one of the addresses reserved by RFC 1918 for private networks:

- ◊ 10.anything
- ◊ 172.x.anything with  $16 \leq x \leq 31$
- ◊ 192.168.anything

These addresses are commonly used on private networks, e.g. behind a Linux machines doing IP masquerade. Machines within the private network can address each other with these addresses. All packets going outside that network, however, have these addresses replaced before they reach the Internet.

If any packets using these addresses do leak out, they do not go far. Most routers automatically discard all such packets.

Various other addresses — the 127.0.0.0/8 block reserved for local use, 0.0.0.0, various broadcast and network addresses — cannot be routed over the Internet, but are not normally included in the meaning when the phrase "non-routable address" is used.

### *NSA*

The US National Security Agency, the American organisation for signals intelligence, the protection of US government messages and the interception and analysis of other messages. For details, see

Bamford's "The Puzzle Palace".

Some history of NSA documents were declassified in response to a FOIA (Freedom of Information Act) request.

*O*

*Oakley*

A key determination protocol, defined in RFC 2412.

*Oakley groups*

The groups used as the basis of Diffie–Hellman key exchange in the Oakley protocol, and in IKE. Four were defined in the original RFC, and a fifth has been added since.

Linux FreeS/WAN currently supports the three groups based on finite fields modulo a prime (Groups 1, 2 and 5) and does not support the elliptic curve groups (3 and 4). For a description of the difference of the types, see discrete logarithms.

*One time pad*

A cipher in which the key is:

- ◊ as long as the total set of messages to be enciphered
- ◊ absolutely random
- ◊ never re–used

Given those three conditions, it can easily be proved that the cipher is perfectly secure, in the sense that an attacker with intercepted message in hand has no better chance of guessing the message than an attacker who has not intercepted the message and only knows the message length. No such proof exists for any other cipher.

There are, however, several problems with this "perfect" cipher.

First, it is **wildly impractical** for most applications. Key management is at best difficult, often completely impossible.

Second, it is **extremely fragile**. Small changes which violate the conditions listed above do not just weaken the cipher little. Quite often they destroy its security completely.

- ◊ Re–using the pad weakens the cipher to the point where it can be broken with pencil and paper. With a computer, the attack is trivially easy.
- ◊ Using *anything* less than truly random numbers *completely* invalidates the security proof.
- ◊ In particular, using computer–generated pseudo–random numbers may give an extremely weak cipher. It might also produce a good stream cipher, if the pseudo–random generator is both well–designed and properly seeded.

Marketing claims about the "unbreakable" security of various products which somewhat resemble one–time pads are common. Such claims are one of the surest signs of cryptographic snake oil; most systems marketed with such claims are worthless.

Finally, even if the system is implemented and used correctly, it is **highly vulnerable to a substitution attack**. If an attacker knows some plaintext and has an intercepted message, he can discover the pad.

- ◊ This does not matter if the attacker is just a passive eavesdropper. It gives him no plaintext he didn't already know and we don't care that he learns a pad which we will never re–use.
- ◊ However, an active attacker who knows the plaintext can recover the pad, then use it to encode with whatever he chooses. If he can get his version delivered instead of yours, this may be a disaster. If you send "attack at dawn", the delivered message can be anything the same length — perhaps "retreat to east" or "shoot generals".

- ◇ An active attacker with only a reasonable guess at the plaintext can try the same attack. If the guess is correct, this works and the attacker's bogus message is delivered. If the guess is wrong, a garbled message is delivered.

In general then, despite its theoretical perfection, the one-time-pad has very limited practical application.

See also the [one time pad FAQ](#).

### *Opportunistic encryption (OE)*

A situation in which any two IPsec-aware machines can secure their communications, without a pre-shared secret and without a common [PKI](#) or previous exchange of public keys. This is one of the goals of the Linux FreeS/WAN project, discussed in our [introduction](#) section.

Setting up for opportunistic encryption is described in our [quickstart](#) document.

### *Opportunistic responder*

A host which accepts, but does not initiate, requests for [opportunistic encryption](#) (OE). An opportunistic responder has enabled OE in its [passive](#) form (pOE) only. A web server or file server may be usefully set up as an opportunistic responder.

Configuring passive OE is described in our [policy groups](#) document.

### *Orange book*

the most basic and best known of the US government's [Rainbow Book](#) series of computer security standards.

### *P*

#### *P1363 standard*

An [IEEE](#) standard for public key cryptography. [Web page](#).

### *pOE*

See [Passive opportunistic encryption](#).

### *Passive attack*

An attack in which the attacker only eavesdrops and attempts to analyse intercepted messages, as opposed to an [active attack](#) in which he diverts messages or generates his own.

### *Passive opportunistic encryption (pOE)*

A form of [opportunistic encryption](#) (OE) in which the host will accept opportunistic connection requests, but will not initiate such requests. A host which runs OE in its passive form only is known as an [opportunistic responder](#).

Configuring passive OE is described in our [policy groups](#) document.

### *Path MTU discovery*

The process of discovering the largest packet size which all links on a path can handle without fragmentation — that is, without any router having to break the packet up into smaller pieces to match the [MTU](#) of its outgoing link.

This is done as follows:

- ◇ originator sends the largest packets allowed by [MTU](#) of the first link, setting the DF (*don't fragment*) bit in the packet header
- ◇ any router which cannot send the packet on (outgoing MTU is too small for it, and DF prevents fragmenting it to match) sends back an [ICMP](#) packet reporting the problem
- ◇ originator looks at ICMP message and tries a smaller size
- ◇ eventually, you settle on a size that can pass all routers
- ◇ thereafter, originator just sends that size and no-one has to fragment

## Introduction to FreeS/WAN

Since this requires co-operation of many systems, and since the next packet may travel a different path, this is one of the trickier areas of IP programming. Bugs that have shown up over the years have included:

- ◊ malformed ICMP messages
- ◊ hosts that ignore or mishandle these ICMP messages
- ◊ firewalls blocking the ICMP messages so host does not see them

Since IPsec adds a header, it increases packet size and may require fragmentation even where incoming and outgoing MTU are equal.

### *Perfect forward secrecy (PFS)*

A property of systems such as Diffie-Hellman key exchange which use a long-term key (such as the shared secret in IKE) and generate short-term keys as required. If an attacker who acquires the long-term key *provably* can

- ◊ *neither* read previous messages which he may have archived
- ◊ *nor* read future messages without performing additional successful attacks

then the system has PFS. The attacker needs the short-term keys in order to read the traffic and merely having the long-term key does not allow him to infer those. Of course, it may allow him to conduct another attack (such as man-in-the-middle) which gives him some short-term keys, but he does not automatically get them just by acquiring the long-term key.

See also Phil Karn's definition.

### *PFS*

see Perfect Forward Secrecy

### *PGP*

**Pretty Good Privacy**, a personal encryption system for email based on public key technology, written by Phil Zimmerman.

The 2.xx versions of PGP used the RSA public key algorithm and used IDEA as the symmetric cipher. These versions are described in RFC 1991 and in Garfinkel's book. Since version 5, the products from PGP Inc. have used Diffie-Hellman public key methods and CAST-128 symmetric encryption. These can verify signatures from the 2.xx versions, but cannot exchange encrypted messages with them.

An IETF working group has issued RFC 2440 for an "Open PGP" standard, similar to the 5.x versions. PGP Inc. staff were among the authors. A free Gnu Privacy Guard based on that standard is now available.

For more information on PGP, including how to obtain it, see our cryptography links.

### *PGP Inc.*

A company founded by Zimmerman, the author of PGP, now a division of NAI. See the corporate website. Zimmerman left in 2001, and early in 2002 NAI announced that they would no longer sell PGP..

Versions 6.5 and later of the PGP product include PGPnet, an IPsec client for Macintosh or for Windows 95/98/NT. See our interoperation document.

### *Photuris*

Another key negotiation protocol, an alternative to IKE, described in RFCs 2522 and 2523.

### *PPP*

**Point-to-Point Protocol**, originally a method of connecting over modems or serial lines, but see also PPPoE.

### *PPPoE*

**PPP over Ethernet**, a somewhat odd protocol that makes Ethernet look like a point-to-point serial link. It is widely used for cable or ADSL Internet services, apparently mainly because it lets the providers use access control and address assignment mechanisms developed for dialup networks. Roaring Penguin provide a widely used Linux implementation.

### *PPTP*

**Point-to-Point Tunneling Protocol**, used in some Microsoft VPN implementations. Papers discussing weaknesses in it are on counterpane.com. It is now largely obsolete, replaced by L2TP.

### *PKI*

**Public Key Infrastructure**, the things an organisation or community needs to set up in order to make public key cryptographic technology a standard part of their operating procedures.

There are several PKI products on the market. Typically they use a hierarchy of Certification Authorities (CAs). Often they use LDAP access to X.509 directories to implement this.

See Web of Trust for a different sort of infrastructure.

### *PKIX*

**PKI eXchange**, an IETF standard that allows PKIs to talk to each other.

This is required, for example, when users of a corporate PKI need to communicate with people at client, supplier or government organisations, any of which may have a different PKI in place. I should be able to talk to you securely whenever:

- ◇ your organisation and mine each have a PKI in place
- ◇ you and I are each set up to use those PKIs
- ◇ the two PKIs speak PKIX
- ◇ the configuration allows the conversation

At time of writing (March 1999), this is not yet widely implemented but is under quite active development by several groups.

### *Plaintext*

The unencrypted input to a cipher, as opposed to the encrypted ciphertext output.

### *Pluto*

The Linux FreeS/WAN daemon which handles key exchange via the IKE protocol, connection negotiation, and other higher-level tasks. Pluto calls the KLIPS kernel code as required. For details, see the manual page `ipsec_pluto(8)`.

### *Public Key Cryptography*

In public key cryptography, keys are created in matched pairs. Encrypt with one half of a pair and only the matching other half can decrypt it. This contrasts with symmetric or secret key cryptography in which a single key known to both parties is used for both encryption and decryption.

One half of each pair, called the public key, is made public. The other half, called the private key, is kept secret. Messages can then be sent by anyone who knows the public key to the holder of the private key. Encrypt with the public key and you know that only someone with the matching private key can decrypt.

Public key techniques can be used to create digital signatures and to deal with key management issues, perhaps the hardest part of effective deployment of symmetric ciphers. The resulting hybrid cryptosystems use public key methods to manage keys for symmetric ciphers.

Many organisations are currently creating PKIs, public key infrastructures to make these benefits widely available.

### *Public Key Infrastructure*

see [PKI](#)

*Q*

*R*

*Rainbow books*

A set of US government standards for evaluation of "trusted computer systems", of which the best known was the [Orange Book](#). One fairly often hears references to "C2 security" or a product "evaluated at B1". The Rainbow books define the standards referred to in those comments.

See this [reference page](#).

The Rainbow books are now mainly obsolete, replaced by the international [Common Criteria](#) standards.

*Random*

A remarkably tricky term, far too much so for me to attempt a definition here. Quite a few cryptosystems have been broken via attacks on weak random number generators, even when the rest of the system was sound.

See [RFC 1750](#) for the theory.

See the manual pages for [ipsec\\_ranbits\(8\)](#) and [ipsec\\_prng\(3\)](#) for more on FreeS/WAN's use of randomness. Both depend on the random(4) device driver..

A couple of years ago, there was extensive mailing list discussion (archived [here](#)) of Linux /dev/random and FreeS/WAN. Since then, the design of the random(4) driver has changed considerably. Linux 2.4 kernels have the new driver..

*Raptor*

A firewall product for Windows NT offering IPsec-based VPN services. Linux FreeS/WAN interoperates with Raptor; see our [interop](#) document for details. Raptor have recently merged with Axent.

*RC4*

**R**ivest **C**ipher four, designed by Ron Rivest of [RSA](#) and widely used. Believed highly secure with adequate key length, but often implemented with inadequate key length to comply with export restrictions.

*RC6*

**R**ivest **C**ipher six, [RSA](#)'s [AES](#) candidate cipher.

*Replay attack*

An attack in which the attacker records data and later replays it in an attempt to deceive the recipient.

*Reverse map*

In [DNS](#), a table where IP addresses can be used as the key for lookups which return a system name and/or other information.

*RFC*

**R**equ**e**st **F**or **C**omments, an Internet document. Some RFCs are just informative. Others are standards.

Our list of [IPsec](#) and other security-related RFCs is [here](#), along with information on methods of obtaining them.

*Rijndael*

a [block cipher](#) designed by two Belgian cryptographers, winner of the US government's [AES](#) contest to pick a replacement for [DES](#). See the [Rijndael home page](#).

*RIPEMD*

A [message digest](#) algorithm. The current version is RIPEMD-160 which gives a 160-bit hash.

*Root CA*

## Introduction to FreeS/WAN

The top level Certification Authority in a hierarchy of such authorities.

### *Routable IP address*

Most IP addresses can be used as "to" and "from" addresses in packet headers. These are the routable addresses; we expect routing to be possible for them. If we send a packet to one of them, we expect (in most cases; there are various complications) that it will be delivered if the address is in use and will cause an ICMP error packet to come back to us if not.

There are also several classes of non-routable IP addresses.

### *RSA algorithm*

**R**ivest **S**hamir **A**dleman public key algorithm, named for its three inventors. It is widely used and likely to become more so since it became free of patent encumbrances in September 2000.

RSA can be used to provide either encryption or digital signatures. In IPsec, it is used only for signatures. These provide gateway-to-gateway authentication for IKE negotiations.

For a full explanation of the algorithm, consult one of the standard references such as Applied Cryptography. A simple explanation is:

The great 17th century French mathematician Fermat proved that,

for any prime  $p$  and number  $x$ ,  $0 \leq x < p$ :

$$\begin{aligned} x^p &= x && \text{modulo } p \\ x^{(p-1)} &= 1 && \text{modulo } p, \text{ non-zero } x \end{aligned}$$

From this it follows that if we have a pair of primes  $p, q$  and two numbers  $e, d$  such that:

$$ed = 1 \quad \text{modulo } \text{lcm}(p-1, q-1)$$

where  $\text{lcm}()$  is least common multiple, then  
for all  $x$ ,  $0 \leq x < pq$ :

$$x^{ed} = x \quad \text{modulo } pq$$

So we construct such a set of numbers  $p, q, e, d$  and publish the product  $N=pq$  and  $e$  as the public key. Using  $c$  for ciphertext and  $i$  for the input plaintext, encryption is then:

$$c = i^e \quad \text{modulo } N$$

An attacker cannot deduce  $i$  from the ciphertext  $c$ , short of either factoring  $N$  or solving the discrete logarithm problem for this field. If  $p, q$  are large primes (hundreds or thousands of bits) no efficient solution to either problem is known.

The receiver, knowing the private key ( $N$  and  $d$ ), can readily recover the plaintext  $p$  since:

$$\begin{aligned} c^d &= (i^e)^d && \text{modulo } N \\ &= i^{ed} && \text{modulo } N \\ &= i && \text{modulo } N \end{aligned}$$

## Introduction to FreeS/WAN

This gives an effective public key technique, with only a couple of problems. It uses a good deal of computer time, since calculations with large integers are not cheap, and there is no proof it is necessarily secure since no-one has proven either factoring or discrete log cannot be done efficiently. Quite a few good mathematicians have tried both problems, and no-one has announced success, but there is no proof they are insoluble.

### *RSA Data Security*

A company founded by the inventors of the RSA public key algorithm.

### *S*

### *SA*

**S**ecurity **A**ssociation, the channel negotiated by the higher levels of an IPsec implementation (IKE) and used by the lower (ESP and AH). SAs are unidirectional; you need a pair of them for two-way communication.

An SA is defined by three things — the destination, the protocol (AH or ESP) and the SPI, security parameters index. It is used as an index to look up other things such as session keys and initialisation vectors.

For more detail, see our section on IPsec and/or RFC 2401.

### *SE Linux*

**S**ecurity **E**nhanced Linux, an NSA-funded project to add mandatory access control to Linux. See the project home page.

According to their web pages, this work will include extending mandatory access controls to IPsec tunnels.

Recent versions of SE Linux code use the Linux Security Module interface.

### *Secure DNS*

A version of the DNS or Domain Name Service enhanced with authentication services. This is being designed by the IETF DNS security working group. Check the Internet Software Consortium for information on implementation progress and for the latest version of BIND. Another site has more information.

IPsec can use this plus Diffie–Hellman key exchange to bootstrap itself. This allows opportunistic encryption. Any pair of machines which can authenticate each other via DNS can communicate securely, without either a pre-existing shared secret or a shared PKI.

### *Secret key cryptography*

See symmetric cryptography

### *Security Association*

see SA

### *Security Enhanced Linux*

see SE Linux

### *Sequence number*

A number added to a packet or message which indicates its position in a sequence of packets or messages. This provides some security against replay attacks.

For automatic keying mode, the IPsec RFCs require that the sender generate sequence numbers for each packet, but leave it optional whether the receiver does anything with them.

### *SHA*

### *SHA-1*

**S**ecure **H**ash **A**lgorithm, a message digest algorithm developed by the NSA for use in the Digital Signature standard, FIPS number 186 from NIST. SHA is an improved variant of MD4 producing a

160-bit hash.

SHA is one of two message digest algorithms available in IPsec. The other is MD5. Some people do not trust SHA because it was developed by the NSA. There is, as far as we know, no cryptographic evidence that SHA is untrustworthy, but this does not prevent that view from being strongly held.

The NSA made one small change after the release of the original SHA. They did not give reasons. It may be a defense against some attack they found and do not wish to disclose. Technically the modified algorithm should be called SHA-1, but since it has replaced the original algorithm in nearly all applications, it is generally just referred to as SHA..

*SHA-256*

*SHA-384*

*SHA-512*

Newer variants of SHA designed to match the strength of the 128, 192 and 256-bit keys of AES. The work to break an encryption algorithm's strength by brute force is 2 keylength operations but a birthday attack on a hash needs only  $2 \sqrt{\text{hashlength}}$ , so as a general rule you need a hash twice the size of the key to get similar strength. SHA-256, SHA-384 and SHA-512 are designed to match the 128, 192 and 256-bit key sizes of AES, respectively.

*Signals intelligence (SIGINT)*

Activities of government agencies from various nations aimed at protecting their own communications and reading those of others. Cryptography, cryptanalysis, wiretapping, interception and monitoring of various sorts of signals. The players include the American NSA, British GCHQ and Canadian CSE.

*SKIP*

Simple **K**ey management for **I**nternet **P**rotocols, an alternative to IKE developed by Sun and being marketed by their Internet Commerce Group.

*Snake oil*

Bogus cryptography. See the Snake Oil FAQ or this paper by Schneier.

*SPI*

**S**ecurity **P**arameter **I**ndex, an index used within IPsec to keep connections distinct. A Security Association (SA) is defined by destination, protocol and SPI. Without the SPI, two connections to the same gateway using the same protocol could not be distinguished.

For more detail, see our IPsec section and/or RFC 2401.

*SSH*

Secure **S**hell, an encrypting replacement for the insecure Berkeley commands whose names begin with "r" for "remote": rsh, rlogin, etc.

For more information on SSH, including how to obtain it, see our cryptography links.

*SSH Communications Security*

A company founded by the authors of SSH. Offices are in Finland and California. They have a toolkit for developers of IPsec applications.

*SSL*

Secure Sockets Layer, a set of encryption and authentication services for web browsers, developed by Netscape. Widely used in Internet commerce. Also known as TLS.

*SSLeay*

A free implementation of SSL by Eric Young (eay) and others. Developed in Australia; not subject to US export controls.

*static IP address*

an IP address which is pre-set on the machine itself, as opposed to a dynamic address which is assigned by a DHCP server or obtained as part of the process of establishing a PPP or PPPoE

connection

### *Stream cipher*

A symmetric cipher which produces a stream of output which can be combined (often using XOR or bitwise addition) with the plaintext to produce ciphertext. Contrasts with block cipher.

IPsec does not use stream ciphers. Their main application is link-level encryption, for example of voice, video or data streams on a wire or a radio signal.

### *subnet*

A group of IP addresses which are logically one network, typically (but not always) assigned to a group of physically connected machines. The range of addresses in a subnet is described using a subnet mask. See next entry.

### *subnet mask*

A method of indicating the addresses included in a subnet. Here are two equivalent examples:

◊ 101.101.101.0/24

◊ 101.101.101.0 with mask 255.255.255.0

The '24' is shorthand for a mask with the top 24 bits one and the rest zero. This is exactly the same as 255.255.255.0 which has three all-ones bytes and one all-zeros byte.

These indicate that, for this range of addresses, the top 24 bits are to be treated as naming a network (often referred to as "the 101.101.101.0/24 subnet") while most combinations of the low 8 bits can be used to designate machines on that network. Two addresses are reserved; 101.101.101.0 refers to the subnet rather than a specific machine while 101.101.101.255 is a broadcast address. 1 to 254 are available for machines.

It is common to find subnets arranged in a hierarchy. For example, a large company might have a /16 subnet and allocate /24 subnets within that to departments. An ISP might have a large subnet and allocate /26 subnets (64 addresses, 62 usable) to business customers and /29 subnets (8 addresses, 6 usable) to residential clients.

### *S/WAN*

Secure Wide Area Network, a project involving RSA Data Security and a number of other companies. The goal was to ensure that all their IPsec implementations would interoperate so that their customers can communicate with each other securely.

### *Symmetric cryptography*

Symmetric cryptography, also referred to as conventional or secret key cryptography, relies on a *shared secret key*, identical for sender and receiver. Sender encrypts with that key, receiver decrypts with it. The idea is that an eavesdropper without the key be unable to read the messages. There are two main types of symmetric cipher, block ciphers and stream ciphers.

Symmetric cryptography contrasts with public key or asymmetric systems where the two players use different keys.

The great difficulty in symmetric cryptography is, of course, key management. Sender and receiver *must* have identical keys and those keys *must* be kept secret from everyone else. Not too much of a problem if only two people are involved and they can conveniently meet privately or employ a trusted courier. Quite a problem, though, in other circumstances.

It gets much worse if there are many people. An application might be written to use only one key for communication among 100 people, for example, but there would be serious problems. Do you actually trust all of them that much? Do they trust each other that much? Should they? What is at risk if that key is compromised? How are you going to distribute that key to everyone without risking its secrecy? What do you do when one of them leaves the company? Will you even know?

## Introduction to FreeS/WAN

On the other hand, if you need unique keys for every possible connection between a group of 100, then each user must have 99 keys. You need either  $99 \times 100 / 2 = 4950$  *secure* key exchanges between users or a central authority that *securely* distributes 100 key packets, each with a different set of 99 keys.

Either of these is possible, though tricky, for 100 users. Either becomes an administrative nightmare for larger numbers. Moreover, keys *must* be changed regularly, so the problem of key distribution comes up again and again. If you use the same key for many messages then an attacker has more text to work with in an attempt to crack that key. Moreover, one successful crack will give him or her the text of all those messages.

In short, the *hardest part of conventional cryptography is key management*. Today the standard solution is to build a hybrid system using public key techniques to manage keys.

*T*

*TIS*

Trusted Information Systems, a firewall vendor now part of NAI. Their Gauntlet product offers IPsec VPN services. TIS implemented the first version of Secure DNS on a DARPA contract.

*TLS*

Transport Layer Security, a newer name for SSL.

*TOS field*

The *Type Of Service* field in an IP header, used to control quality of service routing.

*Traffic analysis*

Deducing useful intelligence from patterns of message traffic, without breaking codes or reading the messages. In one case during World War II, the British guessed an attack was coming because all German radio traffic stopped. The "radio silence" order, intended to preserve security, actually gave the game away.

In an industrial espionage situation, one might deduce something interesting just by knowing that company A and company B were talking, especially if one were able to tell which departments were involved, or if one already knew that A was looking for acquisitions and B was seeking funds for expansion.

In general, traffic analysis by itself is not very useful. However, in the context of a larger intelligence effort where quite a bit is already known, it can be very useful. When you are solving a complex puzzle, every little bit helps.

IPsec itself does not defend against traffic analysis, but carefully thought out systems using IPsec can provide at least partial protection. In particular, one might want to encrypt more traffic than was strictly necessary, route things in odd ways, or even encrypt dummy packets, to confuse the analyst. We discuss this here.

*Transport mode*

An IPsec application in which the IPsec gateway is the destination of the protected packets, a machine acts as its own gateway. Contrast with tunnel mode.

*Triple DES*

see 3DES

*TTL*

*Time To Live*, used to control DNS caching. Servers discard cached records whose TTL expires

*Tunnel mode*

An IPsec application in which an IPsec gateway provides protection for packets to and from other systems. Contrast with transport mode.

*Two-key Triple DES*

## Introduction to FreeS/WAN

A variant of triple DES or 3DES in which only two keys are used. As in the three-key version, the order of operations is EDE or encrypt-decrypt-encrypt, but in the two-key variant the first and third keys are the same.

3DES with three keys has  $3 \times 56 = 168$  bits of key but has only 112-bit strength against a meet-in-the-middle attack, so it is possible that the two key version is just as strong. Last I looked, this was an open question in the research literature.

RFC 2451 defines triple DES for IPsec as the three-key variant. The two-key variant should not be used and is not implemented directly in Linux FreeS/WAN. It cannot be used in automatically keyed mode without major fiddles in the source code. For manually keyed connections, you could make Linux FreeS/WAN talk to a two-key implementation by setting two keys the same in `/etc/ipsec.conf`.

U

V

### *Virtual Interface*

A Linux feature which allows one physical network interface to have two or more IP addresses. See the *Linux Network Administrator's Guide* in book form or on the web for details.

### *Virtual Private Network*

see VPN

### *VPN*

**Virtual Private Network**, a network which can safely be used as if it were private, even though some of its communication uses insecure connections. All traffic on those connections is encrypted.

IPsec is not the only technique available for building VPNs, but it is the only method defined by RFCs and supported by many vendors. VPNs are by no means the only thing you can do with IPsec, but they may be the most important application for many users.

### *VPNC*

Virtual Private Network Consortium, an association of vendors of VPN products.

W

### *Wassenaar Arrangement*

An international agreement restricting export of munitions and other tools of war. Unfortunately, cryptographic software is also restricted under the current version of the agreement. Discussion.

### *Web of Trust*

PGP's method of certifying keys. Any user can sign a key; you decide which signatures or combinations of signatures to accept as certification. This contrasts with the hierarchy of CAs (Certification Authorities) used in many PKIs (Public Key Infrastructures).

See Global Trust Register for an interesting addition to the web of trust.

### *WEP (Wired Equivalent Privacy)*

The cryptographic part of the IEEE standard for wireless LANs. As the name suggests, this is designed to be only as secure as a normal wired ethernet. Anyone with a network connection can tap it. Its advocates would claim this is good design, refusing to build in complex features beyond the actual requirements.

Critics refer to WEP as "Wiretap Equivalent Privacy", and consider it a horribly flawed design based on bogus "requirements". You do not control radio waves as you might control your wires, so the metaphor in the rationale is utterly inapplicable. A security policy that chooses not to invest resources in protecting against certain attacks which can only be conducted by people physically plugged into your LAN may or may not be reasonable. The same policy is completely unreasonable when someone can "plug in" from a laptop half a block away..

## Introduction to FreeS/WAN

There has been considerable analysis indicating that WEP is seriously flawed. A FAQ on attacks against WEP is available. Part of it reads:

... attacks are practical to mount using only inexpensive off-the-shelf equipment. We recommend that anyone using an 802.11 wireless network not rely on WEP for security, and employ other security measures to protect their wireless network. Note that our attacks apply to both 40-bit and the so-called 128-bit versions of WEP equally well.

WEP appears to be yet another instance of governments, and unfortunately some vendors and standards bodies, deliberately promoting hopelessly flawed "security" products, apparently mainly for the benefit of eavesdropping agencies. See this [discussion](#).

X

X.509

A standard from the [ITU \(International Telecommunication Union\)](#), for hierarchical directories with authentication services, used in many [PKI](#) implementations.

Use of X.509 services, via the [LDAP protocol](#), for certification of keys is allowed but not required by the [IPsec RFCs](#). It is not yet implemented in [Linux FreeS/WAN](#).

Xedia

A vendor of router and Internet access products, now part of Lucent. Their QVPN products interoperate with Linux FreeS/WAN; see our [interop document](#).

Y

Z

# Bibliography for the Linux FreeS/WAN project

For extensive bibliographic links, see the [Collection of Computer Science Bibliographies](#)

See our [web links](#) for material available online.

---

Carlisle Adams and Steve Lloyd *Understanding Public Key Infrastructure*  
Macmillan 1999 ISBN 1-57870-166-x

An overview, mainly concentrating on policy and strategic issues rather than the technical details. Both authors work for [PKI](#) vendor [Entrust](#).

---

Albitz, Liu & Loukides *DNS & BIND* 3rd edition  
O'Reilly 1998 ISBN 1-56592-512-2

The standard reference on the [Domain Name Service](#) and [Berkeley Internet Name Daemon](#).

---

Ross Anderson, *Security Engineering – a Guide to Building Dependable Distributed Systems*  
Wiley, 2001, ISBN 0471389226

Easily the best book for the security professional I have seen. **Highly recommended**. See the [book web page](#).

This is quite readable, but Schneier's [Secrets and Lies](#) might be an easier introduction.

---

Bamford *The Puzzle Palace, A report on NSA, Americas's most Secret Agency*  
Houghton Mifflin 1982 ISBN 0-395-31286-8

---

Bamford *Body of Secrets*

The sequel.

---

David Bander, *Linux Security Toolkit*  
IDG Books, 2000, ISBN: 0764546902

This book has a short section on FreeS/WAN and includes Caldera Linux on CD.

---

Chapman, Zwicky & Russell, *Building Internet Firewalls*  
O'Reilly 1995 ISBN 1-56592-124-0

---

Cheswick and Bellovin *Firewalls and Internet Security: Repelling the Wily Hacker*  
Addison-Wesley 1994 ISBN 0201633574

A fine book on firewalls in particular and security in general from two of AT&T's system administrators.

Bellovin has also done a number of [papers](#) on IPsec and co-authored a [paper](#) on a large FreeS/WAN application.

---

Comer *Internetworking with TCP/IP*  
Prentice Hall

## Introduction to FreeS/WAN

- Vol. I: Principles, Protocols, & Architecture, 3rd Ed. 1995 ISBN:0-13-216987-8
- Vol. II: Design, Implementation, & Internals, 2nd Ed. 1994 ISBN:0-13-125527-4
- Vol. III: Client/Server Programming & Applications
  - ◆ AT&T TLI Version 1994 ISBN:0-13-474230-3
  - ◆ BSD Socket Version 1996 ISBN:0-13-260969-X
  - ◆ Windows Sockets Version 1997 ISBN:0-13-848714-6

If you need to deal with the details of the network protocols, read either this series or the [Stevens and Wright](#) series before you start reading the RFCs.

---

Diffie and Landau *Privacy on the Line: The Politics of Wiretapping and Encryption*  
MIT press 1998 ISBN 0-262-04167-7 (hardcover) or 0-262-54100-9

---

Doraswamy and Harkins *IP Sec: The New Security Standard for the Internet, Intranets and Virtual Private Networks*  
Prentice Hall 1999 ISBN: 0130118982

---

Electronic Frontier Foundation *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*  
O'Reilly 1998 ISBN 1-56592-520-3

---

To conclusively demonstrate that DES is inadequate for continued use, the [EFF](#) built a machine for just over \$200,000 that breaks DES encryption in under five days on average, under nine in the worst case.

The book provides details of their design and, perhaps even more important, discusses why they felt the project was necessary. Recommended for anyone interested in any of the three topics mentioned in the subtitle.

See also the [EFF page on this project](#) and our discussion of [DES insecurity](#).

---

Martin Freiss *Protecting Networks with SATAN*  
O'Reilly 1998 ISBN 1-56592-425-8  
translated from a 1996 work in German

SATAN is a Security Administrator's Tool for Analysing Networks. This book is a tutorial in its use.

---

Gaidosch and Kunzinger *A Guide to Virtual Private Networks*  
Prentice Hall 1999 ISBN: 0130839647

---

Simson Garfinkel *Database Nation: the death of privacy in the 21st century*  
O'Reilly 2000 ISBN 1-56592-653-6

A thoughtful and rather scary book.

---

Simson Garfinkel *PGP: Pretty Good Privacy*  
O'Reilly 1995 ISBN 1-56592-098-8

An excellent introduction and user manual for the [PGP](#) email-encryption package. PGP is a good package with a complex and poorly-designed user interface. This book or one like it is a must for anyone who has to use it at length.

## Introduction to FreeS/WAN

The book covers using PGP in Unix, PC and Macintosh environments, plus considerable background material on both the technical and political issues around cryptography.

The book is now seriously out of date. It does not cover recent developments such as commercial versions since PGP 5, the Open PGP standard or GNU PG..

---

Garfinkel and Spafford *Practical Unix Security*  
O'Reilly 1996 ISBN 1-56592-148-8

A standard reference.

Spafford's web page has an excellent collection of [crypto and security links](#).

---

David Kahn *The Codebreakers: the Comprehensive History of Secret Communications from Ancient Times to the Internet*  
second edition Scribner 1996 ISBN 0684831309

A history of codes and code-breaking from ancient Egypt to the 20th century. Well-written and exhaustively researched. **Highly recommended**, even though it does not have much on computer cryptography.

---

David Kahn *Seizing the Enigma, The Race to Break the German U-Boat codes, 1939-1943*  
Houghton Mifflin 1991 ISBN 0-395-42739-8

---

Olaf Kirch *Linux Network Administrator's Guide*  
O'Reilly 1995 ISBN 1-56592-087-2

Now becoming somewhat dated in places, but still a good introductory book and general reference.

---

Kolesnikov and Hatch, *Building Linux Virtual Private Networks (VPNs)*  
New Riders 2002

This has had a number of favorable reviews, including [this one](#) on Slashdot. The book has a [web site](#).

---

Pete Loshin *Big Book of IPsec RFCs*  
Morgan Kaufmann 2000 ISBN: 0-12-455839-9

---

Steven Levy *Crypto: How the Code Rebels Beat the Government -- Saving Privacy in the Digital Age*  
Penguin 2001, ISBN 0-670--85950-8

**Highly recommended.** A fine history of recent (about 1970-2000) developments in the field, and the related political controversies. FreeS/WAN project founder and leader John Gilmore appears several times.

The book does not cover IPsec or FreeS/WAN, but this project is very much another battle in the same war. See our discussion of the [politics](#).

---

Matyas, Anderson et al. *The Global Trust Register*  
Northgate Consultants Ltd 1998 ISBN: 0953239705  
hard cover edition MIT Press 1999 ISBN 0262511053

From their web page:

This book is a register of the fingerprints of the world's most important public keys; it implements a top-level certification authority (CA) using paper and ink rather than in an electronic system.

---

Menezies, van Oorschot and Vanstone *Handbook of Applied Cryptography*  
CRC Press 1997  
ISBN 0-8493-8523-7

An excellent reference. Read Schneier before tackling this.

---

Michael Padlipsky *Elements of Networking Style*  
Prentice-Hall 1985 ISBN 0-13-268111-0 or 0-13-268129-3

Probably ***the funniest technical book ever written***, this is a vicious but well-reasoned attack on the OSI "seven layer model" and all that went with it. Several chapters of it are also available as RFCs 871 to 875.

---

John S. Quarterman *The Matrix: Computer Networks and Conferencing Systems Worldwide*  
Digital Press 1990 ISBN 155558-033-5  
Prentice-Hall ISBN 0-13-565607-9

The best general treatment of computer-mediated communication we have seen. It naturally has much to say about the Internet, but also covers UUCP, Fidonet and so on.

---

David Ranch *Securing Linux Step by Step*  
SANS Institute, 1999

SANS is a respected organisation, this guide is part of a well-known series, and Ranch has previously written the useful Trinity OS guide to securing Linux, so my guess would be this is a pretty good book. I haven't read it yet, so I'm not certain. It can be ordered online from SANS.

Note (Mar 1, 2002): a new edition with different editors in the works. Expect it this year.

---

Bruce Schneier *Applied Cryptography, Second Edition*  
John Wiley & Sons, 1996  
ISBN 0-471-12845-7 hardcover  
ISBN 0-471-11709-9 paperback

A standard reference on computer cryptography. For more recent essays, see the author's company's web site.

---

Bruce Schneier *Secrets and Lies*  
Wiley 2000, ISBN 0-471-25311-1

An interesting discussion of security and privacy issues, written with more of an "executive overview" approach rather than a narrow focus on the technical issues. ***Highly recommended***.

This is worth reading even if you already understand security issues, or think you do. To go deeper, follow it with Anderson's Security Engineering.

---

## Introduction to FreeS/WAN

Scott, Wolfe and Irwin *Virtual Private Networks*  
2nd edition, O'Reilly 1999 ISBN: 1-56592-529-7

This is the only O'Reilly book, out of a dozen I own, that I'm disappointed with. It deals mainly with building VPNs with various proprietary tools — PPTP, SSH, Cisco PIX, ... — and touches only lightly on IPsec-based approaches.

That said, it appears to deal competently with what it does cover and it has readable explanations of many basic VPN and security concepts. It may be exactly what some readers require, even if I find the emphasis unfortunate.

---

Kurt Seifried *Linux Administrator's Security Guide*

Available online from Security Portal. It has fairly extensive coverage of IPsec.

---

Richard E Smith *Internet Cryptography*  
ISBN 0-201-92480-3, Addison Wesley, 1997

See the book's home page

---

Neal Stephenson *Cryptonomicon*  
Hardcover ISBN -380-97346-4, Avon, 1999.

A novel in which cryptography and the net figure prominently. **Highly recommended:** I liked it enough I immediately went out and bought all the author's other books.

There is also a paperback edition. Sequels are expected.

---

Stevens and Wright *TCP/IP Illustrated*  
Addison-Wesley

- Vol. I: The Protocols 1994 ISBN:0-201-63346-9
- Vol. II: The Implementation 1995 ISBN:0-201-63354-X
- Vol. III: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols 1996 ISBN: 0-201-63495-3

If you need to deal with the details of the network protocols, read either this series or the Comer series before you start reading the RFCs.

---

Rubini *Linux Device Drivers*  
O'Reilly & Associates, Inc. 1998 ISBN 1-56592-292-1

---

Robert Zeigler *Linux Firewalls*  
Newriders Publishing, 2000 ISBN 0-7537-0900-9

A good book, with detailed coverage of ipchains(8) firewalls and of many related issues.

# IPsec RFCs and related documents

## The RFCs.tar.gz Distribution File

The Linux FreeS/WAN distribution is available from [our primary distribution site](#) and various mirror sites. To give people more control over their downloads, the RFCs that define IP security are bundled separately in the file RFCs.tar.gz.

The file you are reading is included in the main distribution and is available on the web site. It describes the RFCs included in the [RFCs.tar.gz](#) bundle and gives some pointers to [other ways to get them](#).

## Other sources for RFCs & Internet drafts

### RFCs

RFCs are downloadable at many places around the net such as:

- <http://www.rfc-editor.org>
- [NSF.net](#)
- [Sunsite in the UK](#)

browsable in HTML form at others such as:

- [landfield.com](#)
- [Connected Internet Encyclopedia](#)

and some of them are available in translation:

- [French](#)

There is also a published [Big Book of IPSEC RFCs](#).

### Internet Drafts

Internet Drafts, working documents which sometimes evolve into RFCs, are also available.

- [Overall reference page](#)
- [IPsec](#) working group
- [IPSR \(IPsec Remote Access\)](#) working group
- [IPsec Policy](#) working group
- [KINK \(Kerberized Internet Negotiation of Keys\)](#) working group

Note: some of these may be obsolete, replaced by later drafts or by RFCs.

### FIPS standards

Some things used by [IPsec](#), such as [DES](#) and [SHA](#), are defined by US government standards called [FIPS](#). The issuing organisation, [NIST](#), have a [FIPS home page](#).

## What's in the RFCs.tar.gz bundle?

All filenames are of the form rfc\*.txt, with the \* replaced with the RFC number.

RFC#	Title
------	-------

### Overview RFCs

2401	Security Architecture for the Internet Protocol
2411	IP Security Document Roadmap

### Basic protocols

2402	IP Authentication Header
2406	IP Encapsulating Security Payload (ESP)

### Key management

2367	PF_KEY Key Management API, Version 2
2407	The Internet IP Security Domain of Interpretation for ISAKMP
2408	Internet Security Association and Key Management Protocol (ISAKMP)
2409	The Internet Key Exchange (IKE)
2412	The OAKLEY Key Determination Protocol
2528	Internet X.509 Public Key Infrastructure

### Details of various things used

2085	HMAC-MD5 IP Authentication with Replay Prevention
2104	HMAC: Keyed-Hashing for Message Authentication
2202	Test Cases for HMAC-MD5 and HMAC-SHA-1
2207	RSVP Extensions for IPSEC Data Flows
2403	The Use of HMAC-MD5-96 within ESP and AH
2404	The Use of HMAC-SHA-1-96 within ESP and AH
2405	The ESP DES-CBC Cipher Algorithm With Explicit IV
2410	The NULL Encryption Algorithm and Its Use With IPsec
2451	The ESP CBC-Mode Cipher Algorithms
2521	ICMP Security Failures Messages

### Older RFCs which may be referenced

1321	The MD5 Message-Digest Algorithm
1828	IP Authentication using Keyed MD5
1829	The ESP DES-CBC Transform
1851	The ESP Triple DES Transform
1852	IP Authentication using Keyed SHA

### RFCs for secure DNS service, which IPsec may use

2137	Secure Domain Name System Dynamic Update
2230	Key Exchange Delegation Record for the DNS
2535	Domain Name System Security Extensions
2536	DSA KEYS and SIGs in the Domain Name System (DNS)
2537	RSA/MD5 KEYS and SIGs in the Domain Name System (DNS)
2538	Storing Certificates in the Domain Name System (DNS)
2539	Storage of Diffie-Hellman Keys in the Domain Name System (DNS)

## RFCs labelled "experimental"

2521	ICMP Security Failures Messages
2522	Photuris: Session-Key Management Protocol
2523	Photuris: Extended Schemes and Attributes

## Related RFCs

1750	Randomness Recommendations for Security
1918	Address Allocation for Private Internets
1984	IAB and IESG Statement on Cryptographic Technology and the Internet
2144	The CAST-128 Encryption Algorithm

# Distribution Roadmap: What's Where in Linux FreeS/WAN

This file is a guide to the locations of files within the FreeS/WAN distribution. Everything described here should be on your system once you download, gunzip, and untar the distribution.

This distribution contains two major subsystems

## KLIPS

the kernel code

## Pluto

the user-level key-management daemon

plus assorted odds and ends.

## Top directory

The top directory has essential information in text files:

### *README*

introduction to the software

### *INSTALL*

short experts-only installation procedures. More detailed procedures are in [installation](#) and [configuration](#) HTML documents.

### *BUGS*

major known bugs in the current release.

### *CHANGES*

changes from previous releases

### *CREDITS*

acknowledgement of contributors

### *COPYING*

licensing and distribution information

## Documentation

The doc directory contains the bulk of the documentation, most of it in HTML format. See the [index file](#) for details.

## KLIPS: kernel IP security

KLIPS is **Ke**rnel **L**IP Security. It lives in the klips directory, of course.

### *klips/doc*

documentation

### *klips/patches*

patches for existing kernel files

### *klips/test*

test stuff  
*klips/utils*  
low-level user utilities  
*klips/net/ipsec*  
actual klips kernel files  
*klips/src*  
symbolic link to klips/net/ipsec

The "make insert" step of installation installs the patches and makes a symbolic link from the kernel tree to klips/net/ipsec. The odd name of klips/net/ipsec is dictated by some annoying limitations of the scripts which build the Linux kernel. The symbolic-link business is a bit messy, but all the alternatives are worse.

*klips/utils*  
Utility programs:

*eroute*  
manipulate IPsec extended routing tables  
*klipsdebug*  
set Klips (kernel IPsec support) debug features and level  
*spi*  
manage IPsec Security Associations  
*spigrp*  
group/ungroup IPsec Security Associations  
*tncfg*  
associate IPsec virtual interface with real interface  
These are all normally invoked by ipsec(8) with commands such as

```
ipsec tncfg arguments
```

There are section 8 man pages for all of these; the names have "ipsec\_" as a prefix, so your man command should be something like:

```
man 8 ipsec_tncfg
```

## Pluto key and connection management daemon

Pluto is our key management and negotiation daemon. It lives in the pluto directory, along with its low-level user utility, whack.

There are no subdirectories. Documentation is a man page, pluto.8. This covers whack as well.

## Utils

The utils directory contains a growing collection of higher-level user utilities, the commands that administer and control the software. Most of the things that you will actually have to run yourself are in there.

*ipsec*  
invoke IPsec utilities

## Introduction to FreeS/WAN

ipsec(8) is normally the only program installed in a standard directory, /usr/local/sbin. It is used to invoke the others, both those listed below and the ones in klips/utils mentioned above.

*auto*

control automatically-keyed IPsec connections

*manual*

take manually-keyed IPsec connections up and down

*barf*

generate copious debugging output

*look*

generate moderate amounts of debugging output

There are .8 manual pages for these. look is covered in barf.8. The man pages have an "ipsec\_" prefix so your man command should be something like:

```
man 8 ipsec_auto
```

Examples are in various files with names utils/\*.eg

## Libraries

### FreeS/WAN Library

The lib directory is the FreeS/WAN library, also steadily growing, used by both user-level and kernel code. It includes section 3 man pages for the library routines.

### Imported Libraries

#### LibDES

The libdes library, originally from SSLeay, is used by both Klips and Pluto for Triple DES encryption. Single DES is not used because it is insecure.

Note that this library has its own license, different from the GPL used for other code in FreeS/WAN.

The library includes its own documentation.

#### GMP

The GMP (GNU multi-precision) library is used for multi-precision arithmetic in Pluto's key-exchange code and public key code.

Older versions (up to 1.7) of FreeS/WAN included a copy of this library in the FreeS/WAN distribution.

Since 1.8, we have begun to rely on the system copy of GMP.

# User-Mode-Linux Testing guide

User mode linux is a way to compile a linux kernel such that it can run as a process in another linux system (potentially as a \*BSD or Windows process later). See <http://user-mode-linux.sourceforge.net/>

UML is a good platform for testing and experimenting with FreeS/WAN. It allows several network nodes to be simulated on a single machine. Creating, configuring, installing, monitoring, and controlling these nodes is generally easier and easier to script with UML than real hardware.

You'll need about 500Mb of disk space for a full sunrise-east-west-sunset setup. You can possibly get this down by 130Mb if you remove the sunrise/sunset kernel build. If you just want to run, then you can even remove the east/west kernel build.

Nothing need be done as super user. In a couple of steps, we note where super user is required to install commands in system-wide directories, but ~/bin could be used instead. UML seems to use a system-wide /tmp/uml directory so different users may interfere with one another. Later UMLs use ~/.uml instead, so multiple users running UML tests should not be a problem, but note that a single user running the UML tests will only be able run one set. Further, UMLs sometimes get stuck and hang around. These "zombies" (most will actually be in the "T" state in the process table) will interfere with subsequent tests.

## Preliminary Notes on BIND

As of 2003/3/1, the Light-Weight Resolver is used by pluto. This requires that BIND9 be running. It also requires that BIND9 development libraries be present in the build environment. The DNSSEC code is only truly functional in BIND9 snapshots. The library code could be 9.2.2, we believe. We are using BIND9 20021115 snapshot code from <ftp://ftp.isc.org/isc/bind9/snapshots>.

FreeS/WAN may well require a newer BIND than is on your system. Many distributions have moved to BIND9.2.2 recently due to a security advisory. BIND is five components.

1. named
2. dnssec-\*
3. client side resolver libraries
4. client side utility libraries I thought there were lib and named parts to dnssec...
5. dynamic DNS update utilities

The only piece that we need for \*building\* is #4. That's the only part that has to be on the build host. What is the difference between resolver and util libs? If you want to edit testing/baseconfigs/all/etc/bind, you'll need a snapshot version. The resolver library contains the resolver. FreeS/WAN has its own copy of that in lib/liblwres.

## Steps to Install UML for FreeS/WAN

1. Get the following files:
  - a. from <http://www.sandelman.ottawa.on.ca/freeswan/uml/> umlfreeroot-15.1.tar.gz (or highest numbered one). This is a debian potato root file system. You can use this even on a Redhat host, as it has the newer GLIBC2.2 libraries as well.
  - b. From <ftp://ftp.xs4all.nl/pub/crypto/freeswan/> a snapshot or release (1.92 or better)

## Introduction to FreeS/WAN

- c. From a <http://www.kernel.org/mirror>, the virgin 2.4.19 kernel. Please realize that we have defaults in our tree for kernel configuration. We try to track the latest UML kernels. If you use a newer kernel, you may have faults in the kernel build process. You can see what the latest that is being regularly tested by visiting [freeswan-regress-env.sh](http://freeswan-regress-env.sh).
  - d. Get <http://ftp.nl.linux.org/uml/> uml-patch-2.4.19-47.bz2 or the one associated with your kernel. As of 2003/03/05, uml-patch-2.4.19-47.bz2 works for us. ***More recent versions of the patch have not been tested by us.***
  - e. You'll probably want to visit <http://user-mode-linux.sourceforge.net> and get the UML utilities. These are not needed for the build or interactive use (but recommended). They are necessary for the regression testing procedures used by "make check". We currently use uml\_utilities\_20020212.tar.bz2.
  - f. You need tcpdump version 3.7.1 or better. This is newer than the version included in most LINUX distributions. You can check the version of an installed tcpdump with the --version flag. If you need a newer tcpdump fetch both tcpdump and libpcap source tar files from <http://www.tcpdump.org/> or a mirror.
2. Pick a suitable place, and extract the following files:
- a. 2.4.19 kernel. For instance:

```
cd /c2/kernel
tar xzvf ../download/pub/linux/kernel/v2.4/linux-2.4.19.tar.gz
```

- b. extract the umlfreeroot file

```
mkdir -p /c2/user-mode-linux/basic-root
cd /c2/user-mode-linux/basic-root
tar xzvf ../download/umlfreeroot-15.1.tar.gz
```

- c. FreeSWAN itself (or checkout "all" from CVS)

```
mkdir -p /c2/freeswan/sandbox
cd /c2/freeswan/sandbox
tar xzvf ../download/snapshot.tar.gz
```

3. If you need to build a newer tcpdump:
- ◆ Make sure you have OpenSSL installed — it is needed for cryptographic routines.
  - ◆ Unpack libpcap and tcpdump source in parallel directories (the tcpdump build procedures look for libpcap next door).
  - ◆ Change directory into the libpcap source directory and then build the library:

```
./configure
make
```

- ◆ Change into the tcpdump source directory, build tcpdump, and install it.

```
./configure
make
# Need to be superuser to install in system directories.
# Installing in ~/bin would be an alternative.
su -c "make install"
```

4. If you need the uml utilities, unpack them somewhere then build and install them:

```
cd tools
make all
# Need to be superuser to install in system directories.
```

## Introduction to FreeS/WAN

```
# Installing in ~/bin would be an alternative.
su -c "make install BIN_DIR=/usr/local/bin"
```

### 5. set up the configuration file

- ◆ `cd /c2/freeswan/sandbox/freeswan-1.97/testing/utils`
- ◆ copy `umlsetup-sample.sh` to `../.. /umlsetup.sh`: `cp umlsetup-sample.sh ../.. /umlsetup.sh`
- ◆ open up `../.. /umlsetup.sh` in your favorite editor.
- ◆ change `POOLSPACE=` to point to the place with at least 500Mb of disk. Best if it is on the same partition as the "umlfreeroot" extraction, as it will attempt to use hard links if possible to save disk space.
- ◆ Set `TESTINGROOT` if you intend to run the script outside of the sandbox/snapshot/release directory. Otherwise, it will configure itself.
- ◆ `KERNPOOL` should point to the directory with your 2.4.19 kernel tree. This tree should be unconfigured! This is the directory you used in step 2a.
- ◆ `UMLPATCH` should point at the bz2 file you downloaded at 1d. If using a kernel that already includes the patch, set this to `/dev/null`.
- ◆ `FREESWANDIR` should point at the directory where you unpacked the snapshot/release. Include the "freeswan-snap2001sep16b" or whatever in it. If you are running from CVS, then you point at the directory where `top`, `klips`, etc. are. The script will fix up the directory so that it can be used.
- ◆ `BASICROOT` should be set to the directory used in 2b, or to the directory that you created with RPMs.
- ◆ `SHAREDIR` should be set to the directory used in 2c, to `/usr/share` for Debian potato users, or to `$BASICROOT/usr/share`.

### 6. `cd $TESTINGROOT/utils` `sh make-uml.sh`

It will grind for awhile. If there are errors it will bail. If so, run it under "script" and send the output to [bugs@lists.freeswan.org](mailto:bugs@lists.freeswan.org).

### 7. You will have a bunch of stuff under `$POOLSPACE`. Open four xterms:

```
for i in sunrise sunset east west
do
    xterm -name $i -title $i -e $POOLSPACE/$i/start.sh &    done
```

8. Login as root. Password is "root" (Note, these virtual machines are networked together, but are not configured to talk to the rest of the world.)
9. verify that pluto started on east/west, run "ipsec look"
10. login to sunrise. run "ping sunset"
11. login to west. run "tcpdump -p -i eth1 -n" (tcpdump must be version 3.7.1 or newer)
12. Closing a console xterm will shut down that UML.
13. You can "make check", if you want to. It is run from `/c2/freeswan/sandbox/freeswan-1.97`.

# Debugging the kernel with GDB

With User-Mode-Linux, you can debug the kernel using GDB. See <http://user-mode-linux.sourceforge.net/debugging.html>.

Typically, one will want to address a test case for a failing situation. Running GDB from Emacs, or from other front ends is possible. First start GDB.

Tell it to open the UMLPOOL/swan/linux program.

Note the PID of GDB:

```
marajade-[projects/freeswan/mgmt/planning] mcr 1029 %ps ax | grep gdb
1659 pts/9      SN      0:00 /usr/bin/gdb -fullname -cd /mara4/freeswan/kernpatch/UMLPOOL/swan/ lin
```

Set the following in the environment:

```
UML_east_OPT="debug gdb-pid=1659"
```

Then start the user-mode-linux in the test scheme you wish:

```
marajade-[kernpatch/testing/klips/east-icmp-02] mcr 1220 %../../utils/runme.sh
```

The user-mode-linux will stop on boot, giving you a chance to attach to the process:

```
(gdb) file linux
Reading symbols from linux...done.
(gdb) attach 1
Attaching to program: /mara4/freeswan/kernpatch/UMLPOOL/swan/linux, process 1
0xa0118bc1 in kill () at hostfs_kern.c:770
```

At this point, break points should be created as appropriate.

## Other notes about debugging

If you are running a standard test, after all the packets are sent, the UML will be shutdown. This can cause problems, because the UML may get terminated while you are debugging.

The environment variable NETJIGWAITUSER can be set to "waituser". If so, then the testing system will prompt before exiting the test.

The environment variable UML\_GETTY if set, will cause each UML to spawn a getty on /dev/tty1, and will wait for it to exit before continuing.

# User-Mode-Linux mysteries

- running more than one UML of the same name (e.g. "west") can cause problems.
- running more than one UML from the same root file system is not a good idea.
- all this means that running "make check" twice on the same machine is probably not a good idea.
- occasionally, UMLs will get stuck. This can happen like: 15134 ? T 0:00  
/spare/hugh/uml/uml2.4.18-sept5/umlbuild/east/linux (east) [/bin/sh] 15138 ? T 0:00  
/spare/hugh/uml/uml2.4.18-sept5/umlbuild/east/linux (east) [halt] these will need to be killed. Note that they are in "T" racing mode.
- UMLs can also hang, and will report "Tracing myself and I can't get out". This is a bug in UML. There are ways to find out what is going on and report this to the UML people, but we don't know the magic right now.

## Getting more info from `uml_netjig`

`uml_netjig` can be compiled with a built-in `tcpdump`. This uses not-yet-released code from [www.tcpdump.org](http://www.tcpdump.org). Please see the instructions in `testing/utils/uml_netjig/Makefile`.

# How to configure to use "make check"

## What is "make check"

"make check" is a target in the top level makefile. It takes care of running a number of unit and system tests to confirm that FreeSWAN has been compiled correctly, and that no new bugs have been introduced.

As FreeSWAN contains both kernel and userspace components, doing testing of FreeSWAN requires that the kernel be simulated. This is typically difficult to do as a kernel requires that it be run on bare hardware. A technology has emerged that makes this simpler. This is User Mode Linux.

User-Mode Linux is a way to build a Linux kernel such that it can run as a process under another Linux (or in the future other) kernel. Presently, this can only be done for 2.4 guest kernels. The host kernel can be 2.2 or 2.4.

"make check" expects to be able to build User-Mode Linux kernels with FreeSWAN included. To do this it needs to have some files downloaded and extracted prior to running "make check". This is described in the UML testing document.

After having run the example in the UML testing document and successfully brought up the four machine combination, you are ready to use "make check"

## Running "make check"

"make check" works by walking the FreeSWAN source tree invoking the "check" target at each node. At present there are tests defined only for the `klips` directory. These tests will use the UML infrastructure to test out pieces of the `klips` code.

The results of the tests can be recorded. If the environment variable `$REGRESSRESULTS` is non-null, then the results of each test will be recorded. This can be used as part of a nightly regression testing system, see Nightly testing for more details.

"make check" otherwise prints a minimal amount of output for each test, and indicates pass/fail status of each test as they are run. Failed tests do not cause failure of the target in the form of exit codes.

# How to write a "make check" test

## Structure of a test

Each test consists of a set of directories under `testing/`. There are directories for `klips`, `pluto`, `packaging` and `libraries`. Each directory has a list of tests to run is stored in a file called `TESTLIST` in that directory. e.g. `testing/klips/TESTLIST`.

## The TESTLIST

This isn't actually a shell script. It just looks like one. Some tools other than `/bin/sh` process it. Lines that start with `#` are comments.

```
# test-kind      directory-containing-test      expectation      [PR#]
```

The first word provides the test type, detailed below.

The second word is the name of the test to run. This the directory in which the test case is to be found..

The third word may be one of:

*blank/good*

the test is believed to function, report failure

*bad*

the test is known to fail, report unexpected success

*suspended*

the test should not be run

The fourth word may be a number, which is a PR# if the test is failing.

## Test kinds

The test types are:

*skiptest*

means run no test.

*ctltest*

means run a single system without input/output.

*klipstest*

means run a single system with input/output networks

*umlplutotest*

means run a pair of systems

*umlXhost*

run an arbitrary number of systems

*suntest (TBD)*

means run a quad of east/west/sunrise/sunset

*roadtest (TBD)*

means run a trio of east-sunrise + warrior

*extrudedtest (TBD)*

means run a quad of east-sunrise + warriorsouth + park

*mkinsttest*

a test of the "make install" machinery.

*kernel\_test\_patch*

a test of the "make kernelpatch" machinery.

Tests marked (TBD) have yet to be fully defined.

Each test directory has a file in it called `testparams.sh`. This file sets a number of environment variables to define the parameters of the test.

## Common parameters

*TESTNAME*

the name of the test (repeated for checking purposes)

*TEST\_TYPE*

the type of the test (repeat of type type above)

*TESTHOST*

the name of the UML machine to run for the test, typically "east" or "west"

*TEST\_PURPOSE*

The purpose of the test is one of:

*goal*

The goal purpose is where a test is defined for code that is not yet finished. The test indicates when the goals have in fact been reached.

*regress*

This is a test to determine that a previously existing bug has been repaired. This test will initially be created to reproduce the bug in isolation, and then the bug will be fixed.

*exploit*

This is a set of packets/programs that causes a vulnerability to be exposed. It is a specific variation of the regress option.

*TEST\_GOAL\_ITEM*

in the case of a goal test, this is a reference to the requirements document

*TEST\_PROB\_REPORT*

in the case of regression test, this the problem report number from GNATS

*TEST\_EXPLOIT\_URL*

in the case of an exploit, this is a URL referencing the paper explaining the origin of the test and the origin of exploit software

*REF\_CONSOLE\_OUTPUT*

a file in the test directory that contains the sanitized console output against which to compare the output of the actual test.

*REF\_CONSOLE\_FIXUPS*

a list of scripts (found in `klips/test/fixups`) to apply to sanitize the console output of the machine under test. These are typically perl, awk or sed scripts that remove things in the kernel output that change each time the test is run and/or compiled.

*INIT\_SCRIPT*

a file of commands that is fed into the virtual machine's console in single user mode prior to starting the tests. This file will usually set up any eroute's and SADB entries that are required for the test.

Lines beginning with # are skipped. Blank lines are skipped. Otherwise, a shell prompt is waited for each time (consisting of \n#) and then the command is sent. Note that the prompt is waited for before the command and not after, so completion of the last command in the script is not required. This is often used to invoke a program to monitor the system, e.g. `ipsec pf_key`.

### *RUN\_SCRIPT*

a file of commands that is fed into the virtual machine's console in single user mode, before the packets are sent. On single machine tests, this script doesn't provide any more power than `INIT_SCRIPT`, but is implemented for consistency's sake.

### *FINAL\_SCRIPT*

a file of commands that is fed into the virtual machine's console in single user mode after the final packet is sent. Similar to `INIT_SCRIPT`, above. If not specified, then the single command "halt" is sent. If specified, then the script should end with a halt command to nicely shutdown the UML.

### *CONSOLEDIFFDEBUG*

If set to "true" then the series of console fixups (see `REF_CONSOLE_FIXUPS`) will be output after it is constructed. (It should be set to "false", or unset otherwise)

### *NETJIGDEBUG*

If set to "true" then the series of console fixups (see `REF_CONSOLE_FIXUPS`) will be output after it is constructed. (It should be set to "false", or unset otherwise)

### *NETJIGTESTDEBUG*

If set to "netjig", then the results of talking to the `uml_netjig` will be printed to stderr during the test. In addition, the jig will be invoked with `--debug`, which causes it to log its process ID, and wait 60 seconds before continuing. This can be used if you are trying to debug the `uml_netjig` program itself.

### *HOSTTESTDEBUG*

If set to "hosttest", then the results of talking to the consoles of the UMLs will be printed to stderr during the test.

### *NETJIGWAITUSER*

If set to "waituser", then the scripts will wait forever for user input before they shut the tests down. Use this if you are debugging through the kernel.

### *PACKETRATE*

A number, in milliseconds (default is 500ms) at which packets will be replayed by the netjig.

## KLIPStest paramaters

The `klipstest` function starts a program (`testing/utils/uml_netjig/uml_netjig`) to setup a bunch of I/O sockets (that simulate network interfaces). It then exports the references to these sockets to the environment and invokes (using `system()`) a given script. It waits for the script to finish.

The script invoked (`testing/utils/host-test.tcl`) is a TCL expect script that arranges to start the UML and configure it appropriately for the test. The configuration is done with the script given above for `INIT_SCRIPT`. The TCL script then forks, leaves the UML in the background and exits. `uml_netjig` continues. It then starts listening to the simulated network answering ARPs and inserting packets as appropriate.

The `klipstest` function invokes `uml_netjig` with arguments to capture output from network interface(s) and insert packets as appropriate:

### *PUB\_INPUT*

a pcap file to feed in on the public (encrypted) interface. (typically, `eth1`)

### *PRIV\_INPUT*

a pcap file to feed in on the private (plain-text) interface (typically, eth0).

### *REF\_PUB\_OUTPUT*

a text file containing tcpdump output. Packets on the public (eth1) interface are captured to a pcap file by `uml_netjig`. The `klipstest` function then uses tcpdump on the file to produce text output, which is compared to the file given.

### *REF\_PUB\_FILTER*

a program that will filter the TCPDUMP output to do further processing. Defaults to "cat".

### *REF\_PRIV\_OUTPUT*

a text file containing tcpdump output. Packets on the private (eth0) interface are captured and compared after conversion by tcpdump, as with *REFPUBOUTPUT*.

### *REF\_PRIV\_FILTER*

a program that will filter the TCPDUMP output to do further processing. Defaults to "cat".

### *EXITONEMPTY*

a flag for `uml_netjig`. It should contain "--exitonempty" if `uml_netjig` should exit when all of the input (*PUBINPUT*, *PRIVINPUT*) packets have been injected.

### *ARPREPLY*

a flag for `uml_netjig`. It should contain "--arpreply" if `uml_netjig` should reply to ARP requests. One will typically set this to avoid having to fudge the ARP cache manually.

### *TCPDUMPFLAGS*

a set of flags for the tcpdump used when converting captured output. Typical values will include "-n" to turn off DNS, and often "-E" to set the decryption key (tcpdump 3.7.1 and higher only) for ESP packets. The "-t" flag (turn off timestamps) is provided automatically

### *NETJIG\_EXTRA*

additional comments to be sent to the netjig. This may arrange to record or create additional networks, or may toggle options.

## mkinsttest paramaters

The basic concept of the `mkinsttest` test type is that it performs a "make install" to a temporary `$DESTDIR`. The resulting tree can then be examined to determine if it was done properly. The files can be uninstalled to determine if the file list was correct, or the contents of files can be examined more precisely.

### *INSTALL\_FLAGS*

If set, then an install will be done. This provides the set of flags to provide for the install. The target to be used (usually "install") must be among the flags.

### *POSTINSTALL\_SCRIPT*

If set, a script to run after initial "make install". Two arguments are provided: an absolute path to the root of the FreeSWAN src tree, and an absolute path to the temporary installation area.

### *INSTALL2\_FLAGS*

If set, a second install will be done using these flags. Similarly to *INSTALL\_FLAGS*, the target must be among the flags.

### *UNINSTALL\_FLAGS*

If set, an uninstall will be done using these flags. Similarly to *INSTALL\_FLAGS*, the target (usually "uninstall") must be among the flags.

### *REF\_FIND\_f\_l\_OUTPUT*

If set, a `find $ROOT ( -type f -or -type -l )` will be done to get a list of a real files and symlinks. The resulting file will be compared to the file listed by this option.

### *REF\_FILE\_CONTENTS*

If set, it should point to a file containing records for the form:

```
reffile    samplefile
```

one record per line. A diff between the provided reference file, and the sample file (located in the temporary installation root) will be done for each record.

## **rpm\_build\_install\_test paramaters**

The `rpm_build_install_test` type is to verify that the proper packing list is produced by "make rpm", and that the mechanisms for building the kernel modules produce consistent results.

### *RPM\_KERNEL\_SOURCE*

Point to an extracted copy of the RedHat kernel source code. Variables from the environment may be used.

### *REF\_RPM\_CONTENTS*

This is a file containing one record per line. Each record consists of a RPM name (may contain wildcards) and a filename to compare the contents to. The RPM will be located and a file list will be produced with `rpm2cpio`.

## **libtest paramaters**

The `libtest` test is for testing library routines. The library file is expected to provided an `#ifdef` by the name of *library*. The `libtest` type invokes the C compiler to compile this file, links it against `libfreeswan.a` (to resolve any other dependancies) and runs the test with the `-r` argument to invoke a regression test.

The library test case is expected to do a self-test, exiting with status code 0 if everything is okay, and with non-zero otherwise. A core dump (exit code greater than 128) is noted specifically.

Unlike other tests, there are no subdirectories required, or other parameters to set.

## **umlplutotest paramaters**

The `umlplutotest` function starts a pair of user mode line processes. This is a 2-host version of `umlXhost`. The "EAST" and "WEST" slots are defined.

## **umlXhost parameters**

The `umlXtest` function starts an arbitrary number of user mode line processes.

The script invoked (`testing/utils/Xhost-test.tcl`) is a TCL expect script that arranges to start each UML and configure it appropriately for the test. It then starts listening (using `uml_netjig`) to the simulated network answering ARPs and inserting packets as appropriate.

`umlXtest` has a series of slots, each of which should be filled by a host. The list of slots is controlled by the variable, `XHOST_LIST`. This variable should be set to a space separated list of slots. The former `umlplutotest` is now implemented as a variation of the `umlXhost` test, with `XHOST_LIST="EAST WEST"`.

For each host slot that is defined, a series of variables should be filled in, defining what configuration scripts to use for that host.

## Introduction to FreeS/WAN

The following are used to control the console input and output to the system. Where the string `${host}` is present, the host slot should be filled in. I.e. for the two host system with `XHOST_LIST="EAST WEST"`, then the variables: `EAST_INIT_SCRIPT` and `WEST_INIT_SCRIPT` will exist.

### *`${host}HOST`*

The name of the UML host which will fill this slot

### *`${host}_INIT_SCRIPT`*

a file of commands that is fed into the virtual machine's console in single user mode prior to starting the tests. This file will usually set up any eroute's and SADB entries that are required for the test.

Similar to `INIT_SCRIPT`, above.

### *`${host}_RUN_SCRIPT`*

a file of commands that is fed into the virtual machine's console in single user mode, before the packets are sent. This set of commands is run after all of the virtual machines are initialized. I.e. after `EAST_INIT_SCRIPT` **AND** `WEST_INIT_SCRIPT`. This script can therefore do things that require that all machines are properly configured.

### *`${host}_RUN2_SCRIPT`*

a file of commands that is fed into the virtual machine's console in single user mode, after the packets are sent. This set of commands is run before any of the virtual machines have been shut down. (I.e. before `EAST_FINAL_SCRIPT` **AND** `WEST_FINAL_SCRIPT`.) This script can therefore catch post-activity status reports.

### *`${host}_FINAL_SCRIPT`*

a file of commands that is fed into the virtual machine's console in single user mode after the final packet is sent. Similar to `INIT_SCRIPT`, above. If not specified, then the single command "halt" is sent. Note that when this script is run, the other virtual machines may already have been killed. If specified, then the script should end with a halt command to nicely shutdown the UML.

### *`REF_${host}_CONSOLE_OUTPUT`*

Similar to `REF_CONSOLE_OUTPUT`, above.

Some additional flags apply to all hosts:

### *`REF_CONSOLE_FIXUPS`*

a list of scripts (found in `klips/test/fixups`) to apply to sanitize the console output of the machine under test. These are typically perl, awk or sed scripts that remove things in the kernel output that change each time the test is run and/or compiled.

In addition to input to the console, the networks may have input fed to them:

### *`EAST_INPUT/WEST_INPUT`*

a pcap file to feed in on the private network side of each network. The "EAST" and "WEST" here refer to the networks, not the hosts.

### *`REF_PUB_FILTER`*

a program that will filter the `TCPDUMP` output to do further processing. Defaults to "cat".

### *`REF_EAST_FILTER/REF_WEST_FILTER`*

a program that will filter the `TCPDUMP` output to do further processing. Defaults to "cat".

<

### *`TCPDUMPFLAGS`*

a set of flags for the `tcpdump` used when converting captured output. Typical values will include "-n" to turn off DNS, and often "-E" to set the decryption key (`tcpdump` 3.7.1 and higher only) for ESP packets. The "-t" flag (turn off timestamps) is provided automatically

### *`REF_EAST_OUTPUT/REF_WEST_OUTPUT`*

a text file containing tcpdump output. Packets on the private (eth0) interface are captured and compared after conversion by tcpdump, as with *REF\_PUB\_OUTPUT*.

There are two additional environment variables that may be set on the command line:

*NETJIGVERBOSE=verbose export NETJIGVERBOSE*

If set, then the test output will be "chatty", and let you know what commands it is running, and as packets are sent. Without it set, the output is limited to success/failure messages.

*NETJIGTESTDEBUG=netjig export NETJIGTESTDEBUG*

This will enable debugging of the communication with uml\_netjig, and turn on debugging in this utility. This does not imply NETJIGVERBOSE.

*HOSTTESTDEBUG=hosttest export HOSTTESTDEBUG*

This will show all interactions with the user-mode-linux consoles

## kernel\_patch\_test paramaters

The kernel\_patch\_test function takes some kernel source, copies it with Indir, and then applies the patch as produced by "make kernelpatch".

The following are used to control the input and output to the system:

*KERNEL\_NAME*

the kernel name, typically something like "linus" or "rh"

*KERNEL\_VERSION*

the kernel version number, as in "2.2" or "2.4".

*KERNEL\_\${KERNEL\_NAME}\_\${KERNEL\_VERSION}\_SRC*

This variable should set in the environment, probably in ~/freeswan-regress-env.sh. Examples of this variables would be KERNEL\_LINUS2\_0\_SRC or KERNEL\_RH7\_3\_SRC. This variable should point to an extracted copy of the kernel source in question.

*REF\_PATCH\_OUTPUT*

a copy of the patch output to compare against

*KERNEL\_PATCH\_LEAVE\_SOURCE*

If set to a non-empty string, then the patched kernel source is not removed at the end of the test. This will typically be set in the environment while debugging.

## module\_compile paramaters

The module\_compile test attempts to build the KLIPS module against a given set of kernel source. This is also done by the RPM tests, but in a very specific manner.

There are two variations of this test – one where the kernel either doesn't need to be configured, or is already done, and tests were there is a local configuration file.

Where the kernel doesn't need to be configured, the kernel source that is found is simply used. It may be a RedHat-style kernel, where one can cause it to configure itself via rhconfig.h-style definitions. Or, it may just be a kernel tree that has been configured.

If the variable KERNEL\_CONFIG\_FILE is set, then a new directory is created for the kernel source. It is populated with Indir(1). The referenced file is then copied in as .config, and "make oldconfig" is used to configure the kernel. This resulting kernel is then used as the reference source.

## Introduction to FreeS/WAN

In all cases, the kernel source is found the same was for the kernelpatch test, i.e. via  
KERNEL\_VERSION/KERNEL\_NAME and  
KERNEL\_\${KERNEL\_NAME}\${KERNEL\_VERSION}\_SRC.

Once there is kernel source, the module is compiled using the top-level "make module" target.

The test is considered successful if an executable is found in OUTPUT/module/ipsec.o at the end of the test.

### *KERNEL\_NAME*

the kernel name, typically something like "linus" or "rh"

### *KERNEL\_VERSION*

the kernel version number, as in "2.2" or "2.4".

### *KERNEL\_\${KERNEL\_NAME}\${KERNEL\_VERSION}\_SRC*

This variable should set in the environment, probably in ~/freeswan-regress-env.sh. Examples of this variables would be KERNEL\_LINUX2\_0\_SRC or KERNEL\_RH7\_3\_SRC. This variable should point to an extracted copy of the kernel source in question.

### *KERNEL\_CONFIG\_FILE*

The configuration file for the kernel.

### *KERNEL\_PATCH\_LEAVE\_SOURCE*

If set to a non-empty string, then the configured kernel source is not removed at the end of the test.

This will typically be set in the environment while debugging.

### *MODULE\_DEF\_INCLUDE*

The include file that will be used to configure the KLIPS module, and possibly the kernel source.

# Current pitfalls

*"tcpdump dissector" not available.*

This is a non-fatal warning. If `uml_netjig` is invoked with the `-t` option, then it will attempt to use `tcpdump`'s dissector to decode each packet that it processes. The dissector is presently not available, so this option is normally turned off at compile time. The dissector library will be released with `tcpdump` version 4.0.

# Nightly regression testing

The nightly regression testing system consists of several shell scripts and some perl scripts. The goal is to check out a fresh tree, run "make check" on it, record the results and summarize the results to the team and to the web site.

Output can be found on [adams](#), although the tests are actually run on another project machine.

# How to setup the nightly build

The best way to do nightly testing is to setup a new account. We call the account "build" – you could call it something else, but there may still be some references to ~build in the scripts.

## Files you need to know about

As few files as possible need to be extracted from the source tree – files are run from the source tree whenever possible. However, there are some bootstrap and configuration files that are necessary.

There are 7 files in testing/utils that are involved:

### *nightly-sample.sh*

This is the root of the build process. This file should be copied out of the CVS tree, to \$HOME/bin/nightly.sh of the build account. This file should be invoked from cron.

### *freeswan-regress-env-sample.sh*

This file should be copied to \$HOME/freeswan-regress-env.sh. It should be edited to localize the values. See below.

### *regress-cleanup.pl*

This file needs to be copied to \$HOME/bin/regress-cleanup.pl. It is invoked by the nightly file before doing anything else. It removes previous nights builds in order to free up disk space for the build about to be done.

### *teammail-sample.sh*

A script used to send results email to the "team". This sample script could be copied to \$HOME/bin/teammail.sh. This version will PGP encrypt all the output to the team members. If this script is used, then PGP will have to be properly setup to have the right keys.

### *regress-nightly.sh*

This is the first stage of the nightly build. This stage will call other scripts as appropriate, and will extract the source code from CVS. This script should be copied to \$HOME/bin/regress-nightly.sh

### *regress-stage2.sh*

This is the second stage of the nightly build. It is called in place. It essentially sets up the UML setup in umlsetup.sh, and calls "make check".

### *regress-summarize-results.pl*

This script will summarize the results from the tests to a permanent directory set by \$REGRESSRESULTS. It is invoked from the stage2 nightly script.

### *regress-chart.sh*

This script is called at the end of the build process, and will summarize each night's results (as saved into \$REGRESSRESULTS by regress-summarize-results.pl) as a chart using gnuplot. Note that this requires at least gnuplot 3.7.2.

## Configuring freeswan-regress-env.sh

For more info on KERNPOOL, UMLPATCH, BASICROOT and SHAREDIR, see [User-Mode-Linux testing guide](#).

### *KERNPOOL*

Extract copy of some kernel source to be used for UML builds

### *UMLPATCH*

matching User-Mode-Linux patch.

### *BASICROOT*

the root file system image (see User-Mode-Linux testing guide).

*SHAREDIR*={*BASICROOT*}/usr/share

The /usr/share to use.

### *REGRESSTREE*

A directory in which to store the nightly regression results. Directories will be created by date in this tree.

*TCPDUMP*=tcpdump-3.7.1

The path to the tcpdump to use. This must have crypto compiled in, and must be at least 3.7.1

*KERNEL\_RH7\_2\_SRC*=/a3/kernel\_sources/linux-2.4.9-13/

An extracted copy of the RedHat 7.2. kernel source. If set, then the packaging/rpm-rh72-install-01 test will be run, and an RPM will be built as a test.

*KERNEL\_RH7\_3\_SRC*=/a3/kernel\_sources/rh/linux-2.4.18-5

An extracted copy of the RedHat 7.3. kernel source. If set, then the packaging/rpm-rh73-install-01 test will be run, and an RPM will be built as a test.

*NIGHTLY\_WATCHERS*="userid,userid,userid"

The list of people who should receive nightly output. This is used by teammail.sh

*FAILLINES*=128

How many lines of failed test output to include in the nightly output

*PATH*=\$PATH:/sandel/bin export *PATH*

You can also override the path if necessary here.

*CVSROOT*=:pserver:anoncvs@ip212.xs4net.freeswan.org:/freeswan/MASTER

The CVSROOT to use. This example may work for anonymous CVS, but will be 12 hours behind the primary, and is still experimental

*SNAPSHOTSIGDIR*=\$HOME/snapshot-sig

For the release tools, where to put the generated per-snapshot signature keys

*LASTREL*=1.97

the name of the last release branch (to find the right per-snapshot signature