

strongswan.org

Downloads

Gitweb

|


Home

Projects

Help

Sign in

Register



strongSwan

Search:

Overview

Activity

Roadmap

Issues

Wiki

Repository

Developer Documentation

UserDocumentation »

History

Documentation

IPsec Documentation

Installation Documentation

User Documentation

Developer Documentation

Wiki

Start page

Index by title

Index by date

ipsec.conf

strongSwan's `/etc/ipsec.conf` configuration file consists of three different section types:

- config setup** defines general configuration parameters
- conn <name>** defines a connection
- ca <name>** defines a certification authority

There can be only one **config setup** section but an unlimited number of **conn** and **ca** sections.

All parameters belonging to a section must be indented by at least one space or tab character. The rest of the line after a '#' character is treated as a comment. Comments within a section must also be indented.

A line which contains **include** followed by a file name is replaced by the contents of that file. If the file name is not a full pathname, it is considered to be relative to the directory containing the including file. Such inclusions can be nested. The file name may include wildcards, for example: `include ipsec.*.conf`

Example

```
# /etc/ipsec.conf - strongSwan IPsec configuration file

config setup
    crlcheckinterval=600s
    cachectrls=yes
    strictcrlpolicy=yes
    plutostart=no

ca strongswan #define alternative CRL distribution point
    cacert=strongswanCert.pem
    crluri=http://crl2.strongswan.org/strongswan.crl
    auto=add

conn %default
    keyingtries=1
    keyexchange=ikev2

conn roadwarrior
    left=192.168.0.1
    leftsubnet=10.1.0.0/16
    leftcert=moonCert.pem
    leftid=@moon.strongswan.org
    right=%any
    auto=add
```

IKE and ESP Cipher Suites

- IKEv1 Cipher Suites**
- IKEv2 Cipher Suites**

Powered by Redmine © 2006-2011 Jean-Philippe Lang

http://wiki.strongswan.org/projects/strongswan/wiki/IpsecConf[19.07.2011 5:27:58 PM]

strongswan.org

Downloads

Gitweb

|


Home

Projects

Help

Sign in

Register



strongSwan

Search:

Overview

Activity

Roadmap

Issues

Wiki

Repository

Developer Documentation

UserDocumentation »

History

ipsec.secrets

strongSwan's `/etc/ipsec.secrets` file contains an unlimited number of the following types of secrets:

- RSA** defines an RSA private key
- ECDSA** defines an ECDSA private key
- PSK** defines a pre-shared key
- EAP** defines EAP credentials
- XAUTH** defines XAUTH credentials
- PIN** defines a smartcard PIN

Whitespace at the end of a line is ignored. At the start of a line or after whitespace, `#` and the following text up to the end of the line is treated as a comment.

An **include** directive causes the contents of the named file to be processed before continuing with the current file. The filename can contain wildcards, so every file with a matching name is processed. Includes may be nested to a modest depth (10, currently). If the filename doesn't start with a `/`, the directory containing the current file is prepended to the name.

ID selectors

Each secret can be preceded by a list of optional ID selectors. The two parts are separated by a colon (`:`) that is surrounded by whitespace. If no ID selectors are specified the line must start with a colon.

A selector is an IP address, a Fully Qualified Domain Name, `user@FQDN`, `%any` or `%any6` (other kinds may come). An IP address may be written in the familiar dotted quad form or as a domain name to be looked up when the file is loaded. In many cases it is a bad idea to use domain names because the name server may not be running or may be insecure. To denote a Fully Qualified Domain Name (as opposed to an IP address denoted by its domain name), precede the name with an at sign (`@`).

Matching IDs with selectors is fairly straightforward: they have to be equal. In the case of a *Road Warrior* connection, if an equal match is not found for the Peer's ID, and it is in the form of an IP address, a selector of `%any` will match the peer's IP address if IPV4 and `%any6` will match a the peer's IP address if IPV6. Currently, the obsolete notation `0.0.0.0` may be used in place of `%any`.

When using IKEv1 an additional complexity arises in the case of authentication by preshared secret: the responder will need to look up the secret before the Peer's ID payload has been decoded, so the ID used will be the IP address.

To authenticate a connection between two hosts, the entry that most specifically matches the host and peer IDs is used. An entry with no selectors will match any host and peer. More specifically, an entry with one selector will match a host and peer if the selector matches the host's ID (the peer isn't considered). Still more specifically, an entry with multiple selectors will match a host and peer if the host ID and peer ID each match one of the selectors. If the key is for an asymmetric authentication technique (i.e. a public key system such as RSA), an entry with multiple selectors will match a host and peer even if only the host ID matches a selector (it is presumed that the selectors are all identities of the host). It is acceptable for two entries to be the best match as long as they agree about the secret or private key.

Authentication by preshared secret requires that both systems find the identical secret (the secret is not actually transmitted by the IKE protocol). If both the host and peer appear in the selector list, the same entry will be suitable for both systems so verbatim copying between systems can be used. This naturally extends to larger groups sharing the same secret. Thus multiple-selector entries are best for PSK authentication.

Authentication by public key systems such as RSA requires that each host have its own private key. A host could reasonably use a different private keys for different interfaces and for different peers. But it would not be normal to share entries between systems. Thus thus no-selector and one-selector forms of entry often make sense for public key authentication.

Example

```
# /etc/ipsec.secrets - strongSwan IPsec secrets file
192.168.0.1 %any : PSK "v+NkxY9LLZvwj4qCC2o/gGrWDF2d2ljL"
: RSA moonKey.pem
alice@strongswan.org : EAP "x3.dEhGN"
carol : XAUTH "4iChxLT3"
dave : XAUTH "ryftzG4A"
# get secrets from other files
include ipsec.*.secrets
```

Documentation

I Psec Documentation

Installation Documentation

User Documentation

Developer Documentation

Wiki

Start page


Index by title

Index by date

Powered by Redmine © 2006-2011 Jean-Philippe Lang

http://wiki.strongswan.org/projects/strongswan/wiki/IpsecSecrets[19.07.2011 5:31:22 PM]

strongswan.orgDownloadsGitweb | HomeProjectsHelp



strongSwan

Search:

Overview

Activity

Roadmap

Issues

Wiki

Repository

Developer Documentation

UserDocumentation »

History

ipsec.d

strongSwan's `/etc/ipsec.d/` directory contains various certificate and CRL files that are loaded by the keying daemons pluto and charon. The following subdirectories are currently defined:

- private** contains RSA and ECDSA private key files
- certs** contains X.509 or PGP end entity certificates
- crls** contains certificate revocation lists
- cacerts** contains trustworthy CA certificates
- ocspcerts** contains trustworthy OCSP signer certificates
- aacerts** contains trustworthy authorization authority certificates
- acerts** contains attribute certificates
- reqs** contains PKCS#10 certificate requests

Documentation

IPsec Documentation

Installation Documentation

User Documentation

Developer Documentation

Wiki

Start page

Index by title

Index by date

Powered by Redmine © 2006-2011 Jean-Philippe Lang

http://wiki.strongswan.org/projects/strongswan/wiki/IpsecDirectory[19.07.2011 5:38:40 PM]

strongswan.org

Downloads

Gitweb

|


Home

Projects

Help

Sign in

Register



strongSwan

Search:

Overview

Activity

Roadmap

Issues

Wiki

Repository

Developer Documentation

UserDocumentation »

History

strongswan.conf

Documentation

IPsec Documentation

Installation Documentation

User Documentation

Developer Documentation

Wiki

Start page

Index by title

Index by date

Overview

Each section has a name, followed by C-Style curly brackets defining the sections body. Each section body contains a set of subsections and key/value pairs:

```
settings := (section|keyvalue)*
section  := name { settings }
keyvalue := key = value\n
```

Values must be terminated by a newline. Comments are possible using the #-character, but be careful: The parser implementation is currently limited and does not like braces in comments. Section names and keys may contain any printable character except:

```
. { } # \n \t space
```

An example might look like this:

```
a = b
section-one {
  somevalue = asdf
  subsection {
    othervalue = xxx
  }
  # yei, a comment
  yetanother = zz
}
section-two {
  x = 12
}
```

Indentation is optional, you may use tabs or spaces.

Including files

Version 4.5.1

 introduced the **include** statement which allows to include other files into strongswan.conf, e.g.

```
include /some/path/*.conf
```

If the file name is not an absolute path, it is considered to be relative to the directory of the file containing the include statement. The file name may include shell wildcards. Also, such inclusions can be nested.

Sections loaded from the included files **extend** previously loaded sections; already existing values are **replaced**. It is important to note that settings are added relative to the section the include statement is in.

As an example, the following three files result in the same final config as the one given above:

```
a = b
section-one {
  somevalue = before include
  include include.conf
}
include other.conf
```

include.conf:

```
# settings loaded from this file are added to section-one
# the following replaces the previous value
somevalue = asdf
subsection {
  othervalue = yyy
}
yetaanother = zz
```

other.conf:

```
# this extends section-one and subsection
section-one {
```

http://wiki.strongswan.org/projects/strongswan/wiki/StrongswanConf[19.07.2011 5:38:53 PM]

```
subsection {
    # this replaces the previous value
    othervalue = xxx
}
section-two {
    x = 12
}
```

## Reading values

The config file is read by libstrongswan during library initialization. Values are accessed using a dot-separated section list and a key:  
Accessing **section-one.subsection.othervalue** will return **xxx**.

Have a look at the settings interface (**source:src/libstrongswan/settings.h**) to learn about the details.

## Defined keys

The following keys are currently defined (using dot notation).

Key	Default	Description
<b>charon section</b>		
charon.block_threshold	5	Maximum number of half-open IKE_SAs for a single peer IP
charon.close_ike_on_child_failure	no	Close the IKE_SA if setup of the CHILD_SA along with IKE_AUTH failed
charon.cookie_threshold	10	Number of half-open IKE_SAs that activate the cookie mechanism
charon.dns1		DNS server 1 assigned to peer via configuration payload (CP), see <b>attr plugin</b>
charon.dns2		DNS server 2 assigned to peer via configuration payload (CP)
charon.dos_protection	yes	Enable Denial of Service protection using cookies and aggressiveness checks
charon.filelog		Section to define file loggers, see <b>LoggerConfiguration</b>
charon.flush_auth_cfg	no	
charon.hash_and_url	no	Enable hash and URL support
charon.ignore_routing_tables		A list of routing tables to be excluded from route lookup
charon.ikesa_table_segments	1	Number of exclusively locked segments in the hash table
charon.ikesa_table_size	1	Size of the IKE_SA hash table
charon.inactivity_close_ike	no	Whether to close IKE_SA if the only CHILD_SA closed due to inactivity
charon.install_routes	yes	Install routes into <b>a separate routing table</b> for established IPsec tunnels
charon.install_virtual_ip	yes	Install virtual IP addresses
charon.keep_alive	20s	NAT keep alive interval
charon.load		Plugins to load in IKEv2 charon daemon, see <b>PluginLoad</b>
charon.max_packet	10000	Maximum packet size accepted by charon
charon.multiple_authentication	yes	Enable multiple authentication exchanges (RFC 4739)
charon.nbns1		WINS server 1 assigned to peer via configuration payload (CP), see <b>attr plugin</b>
charon.nbns2		WINS server 2 assigned to peer via configuration payload (CP)
charon.process_route	yes	Process RTM_NEWROUTE and RTM_DELROUTE events
charon.receive_delay	0	Delay for receiving packets, to simulate larger RTT
charon.receive_delay_response	yes	Delay response messages
charon.receive_delay_request	yes	Delay request messages
charon.receive_delay_type	0	Specific IKEv2 message type to delay, 0 for any
charon.retransmit_base	1.8	Base to use for calculating exponential back off, see <b>Retransmission</b>

charon.retransmit_timeout	4.0	Timeout in seconds before sending first retransmit
charon.retransmit_tries	5	Number of times to retransmit a packet before giving up
charon.reuse_ikesa	yes	Initiate CHILD_SA within existing IKE_SAs
charon.routing_table		Numerical <b>routing table</b> to install routes to
charon.routing_table_prio		Priority of the routing table
charon.send_delay	0	Delay for sending packets, to simulate larger RTT
charon.send_delay_response	yes	Delay response messages
charon.send_delay_request	yes	Delay request messages
charon.send_delay_type	0	Specific IKEv2 message type to delay, 0 for any
charon.send_vendor_id	no	Send strongSwan vendor ID payload
charon.syslog		Section to define syslog loggers, see <b>LoggerConfiguration</b>
charon.threads	16	Number of worker threads in charon
<b>charon plugins subsection</b>		
charon.plugins.android.loglevel	1	Loglevel for logging to Android specific logger
charon.plugins.attr		Section to specify arbitrary attributes that are assigned to a peer via configuration payload, see <b>attr plugin</b>
charon.plugins.dhcp.identity_lease	no	Derive user-defined MAC address from hash of IKEv2 identity
charon.plugins.dhcp.server	255.255.255.255	DHCP server unicast or broadcast IP address, see <b>DHCP plugin</b>
charon.plugins.duplicheck.enable	yes	enable loaded <b>duplicheck</b> plugin
charon.plugins.eap-aka.request_identity	yes	
charon.plugins.eap-aka-3gpp2.seq_check		
charon.plugins.eap-gtc.pam_service	login	PAM service to be used for authentication, see <b>EAP-GTC</b>
charon.plugins.eap-peap.fragment_size	1024	Maximum size of an EAP-PEAP packet
charon.plugins.eap-peap.max_message_count	32	Maximum number of processed EAP-PEAP packets
charon.plugins.eap-peap.include_length	no	Include length in non-fragmented EAP-PEAP packets
charon.plugins.eap-peap.phase2_method	mschapv2	Phase2 EAP client authentication method
charon.plugins.eap-peap.phase2_piggyback	no	Phase2 EAP Identity request piggybacked by server onto TLS Finished message
charon.plugins.eap-peap.request_peer_auth	no	Request peer authentication based on a client certificate
charon.plugins.eap-radius.class_group	no	Use the class attribute sent in the RADIUS-Accept message as group membership information, see <b>EapRadius</b>
charon.plugins.eap-radius.eap_start	no	Send EAP-Start instead of EAP-Identity to start RADIUS conversation
charon.plugins.eap-radius.filter_id	no	Use the filter_id attribute sent in the RADIUS-Accept message as group membership if the RADIUS tunnel_type attribute is set to ESP
charon.plugins.eap-radius.id_prefix		Prefix to EAP-Identity, some AAA servers use a IMSI prefix to select the EAP method
charon.plugins.eap-radius.nas_identifier	strongSwan	NAS-Identifier to include in RADIUS messages
charon.plugins.eap-radius.port	1812	Port of RADIUS server (authentication)
charon.plugins.eap-radius.secret		Shared secret between RADIUS and NAS
charon.plugins.eap-radius.server		IP/Hostname of RADIUS server
charon.plugins.eap-radius.servers		Section to specify multiple RADIUS servers, see <b>EapRadius</b>
charon.plugins.eap-radius.sockets	1	Number of sockets (ports) to use, increase for high load
charon.plugins.eap-sim.request_identity	yes	
charon.plugins.eap-simaka-sql.database		
charon.plugins.eap-simaka-sql.remove_used		

charon.plugins.eap-tls.fragment_size	1024	Maximum size of an EAP-TLS packet
charon.plugins.eap-tls.max_message_count	32	Maximum number of processed EAP-TLS packets
charon.plugins.eap-tls.include_length	yes	Include length in non-fragmented EAP-TLS packets
charon.plugins.eap-tnc.fragment_size	50000	Maximum size of an EAP-TNC packet
charon.plugins.eap-tnc.max_message_count	10	Maximum number of processed EAP-TNC packets
charon.plugins.eap-tnc.include_length	yes	Include length in non-fragmented EAP-TM NC packets
charon.plugins.eap-ttls.fragment_size	1024	Maximum size of an EAP-TTLS packet
charon.plugins.eap-ttls.max_message_count	32	Maximum number of processed EAP-TTLS packets
charon.plugins.eap-ttls.include_length	yes	Include length in non-fragmented EAP-TTLS packets
charon.plugins.eap-ttls.phase2_method	md5	Phase2 EAP client authentication method
charon.plugins.eap-ttls.phase2_piggyback	no	Phase2 EAP Identity request piggybacked by server onto TLS Finished message
charon.plugins.eap-ttls.phase2_tnc	no	Start phase2 EAP TNC protocol after successful client authentication
charon.plugins.eap-ttls.request_peer_auth	no	Request peer authentication based on a client certificate
charon.plugins.ha.fifo_interface	yes	
charon.plugins.ha.heartbeat_delay	1000	
charon.plugins.ha.heartbeat_timeout	2100	
charon.plugins.ha.local		
charon.plugins.ha.monitor	yes	
charon.plugins.ha.pools		
charon.plugins.ha.remote		
charon.plugins.ha.resync	yes	
charon.plugins.ha.secret		
charon.plugins.ha.segment_count	1	
charon.plugins.led.activity_led		
charon.plugins.led.blink_time	50	
charon.plugins.kernel-klips.ipsec_dev_count	4	Number of ipsecN devices
charon.plugins.kernel-klips.ipsec_dev_mtu	0	Set MTU of ipsecN device
charon.plugins.load-tester.child_rekey	600	Seconds to start CHILD_SA rekeying after setup
charon.plugins.load-tester.delay	0	Delay between initiations for each thread
charon.plugins.load-tester.delete_after_established	no	Delete an IKE_SA as soon as it has been established
charon.plugins.load-tester.dpd_delay	0	DPD delay to use in load test
charon.plugins.load-tester.dynamic_port	0	Base port to be used for requests (each client uses a different port)
charon.plugins.load-tester.eap_password	default-pwd	EAP secret to use in load test
charon.plugins.load-tester.enable	no	Enable the load testing plugin. Read <b>LoadTests</b> first!
charon.plugins.load-tester.fake_kernel	no	Fake the kernel interface to allow load-testing against self
charon.plugins.load-tester.ike_rekey	0	Seconds to start IKE_SA rekeying after setup
charon.plugins.load-tester.init_limit	0	Global limit of concurrently established SAs during load test
charon.plugins.load-tester.initiators	0	Number of concurrent initiator threads to use in load test
charon.plugins.load-tester.initiator_auth	pubkey	Authentication method(s) the initiator uses
charon.plugins.load-tester.initiator_id		Initiator ID to use in load test
charon.plugins.load-tester.iterations	1	Number of IKE_SAs to initiate to self by each initiator in load test
charon.plugins.load-tester.pool		Provide INTERNAL_IPV4_ADDRs from a named pool

charon.plugins.load-tester.preshared_key	default-psk	Preshared key to use in load test
charon.plugins.load-tester.proposal	aes128-sha1-modp768	IKE proposal to use in load test
charon.plugins.load-tester.remote	127.0.0.1	Address to initiation connections to
charon.plugins.load-tester.responder_auth	pubkey	Authentication method(s) the responder uses
charon.plugins.load-tester.responder_id		Responder ID to use in load test
charon.plugins.load-tester.request_virtual_ip	no	Request an INTERNAL_IPV4_ADDR from the server
charon.plugins.load-tester.shutdown_when_complete	no	Shutdown the daemon after all IKE_SAs have been established
charon.plugins.resolve.file	/etc/resolv.conf	File where to add DNS server entries
charon.plugins.sql.database		Database URI for charons <b>SQL</b> plugin
charon.plugins.sql.loglevel	-1	Loglevel for logging to <b>SQL</b> database
charon.plugins.tnc_imc.preferred_language	en	Preferred language for TNC recommendations
charon.plugins.tnc_imv.tnc_config	/etc/tnc_config	TNC IMC configuration directory
charon.plugins.tnc_imc.tnc_config	/etc/tnc_config	TNC IMV configuration directory
charon.plugins.whitelist.enable	yes	enable loaded <b>whitelist</b> plugin
<b>libstrongswan section</b>		
libstrongswan.crypto_test.bench	no	
libstrongswan.crypto_test.bench_size	1024	
libstrongswan.crypto_test.bench_time	50	
libstrongswan.crypto_test.on_add	no	Test crypto algorithms during registration
libstrongswan.crypto_test.on_create	no	Test crypto algorithms on each crypto primitive instantiation
libstrongswan.crypto_test.required	no	Strictly require at least one test vector to enable an algorithm
libstrongswan.crypto_test.rng_true	no	Whether to test RNG with TRUE quality; requires a lot of entropy
libstrongswan.dh_exponent_ansi_x9_42	yes	Use ANSI X9.42 DH exponent size or optimum size matched to cryptographical strength
libstrongswan.ecp_x_coordinate_only	yes	Compliance with the errata for RFC 4753
libstrongswan.integrity_test	no	Check daemon, libstrongswan and plugin integrity at startup
libstrongswan.leak_detective.detailed	yes	Includes source file names and line numbers in leak detective output
libstrongswan.x509.enforce_critical	yes	Discard certificates with unsupported or unknown critical extensions
<b>libstrongswan plugins subsection</b>		
libstrongswan.plugins.attr-sql.database		Database URI for the <b>attr-sql plugin</b> used by charon and pluto
libstrongswan.plugins.attr-sql.lease_history	yes	Enable logging of <b>SQL</b> IP pool leases
libstrongswan.plugins.gcrypt.quick_random	no	Use faster random numbers in gcrypt; for testing only, produces weak keys!
libstrongswan.plugins.openssl.engine_id	pkcs11	ENGINE ID to use in the OpenSSL plugin
libstrongswan.plugins.pkcs11.modules		List of available PKCS#11 modules, see <b>SmartCardsIKEv2</b>
libstrongswan.plugins.pkcs11.use_hasher	no	Whether the PKCS#11 modules should be used to hash data
<b>libtls section</b>		
libtls.cipher		List of TLS encryption ciphers
libtls.key_exchange		List of TLS key exchange methods
libtls.mac		List of TLS MAC algorithms
libtls.suites		List of TLS cipher suites
<b>libimcv section</b>		
libimcv.debug_level	1	Debug level for a standalone libimcv library
libimcv.stderr_quiet	no	Disable the output to stderr in a standalone libimcv library



<b>libimcv plugins subsection</b>		
libimcv.plugins.imc_test.command	none	Command to be sent to the IMV Test
libimcv.plugins.imc_test.retry	no	Do a handshake retry
libimcv.plugins.imc_test.retry_command		Command to be sent to the IMV Test in the handshake retry
libimcv.plugins.imv_test.rounds	0	Number of IMC-IMV retry rounds
libimcv.plugins.imv_scanner.closed_port_policy	yes	By default all ports must be closed (yes) or can be open (no)
libimcv.plugins.imv_scanner.tcp_ports		List of TCP ports that can be open or must be closed
libimcv.plugins.imv_scanner.udp_ports		List of UDP ports that can be open or must be closed
<b>manager section</b>		
manager.database		Credential database URI for manager
manager.debug	no	Enable debugging in manager
manager.load		Plugins to load in manager
manager.socket		FastCGI socket of manager, to run it statically
manager.threads	10	Threads to use for request handling
manager.timeout	15m	Session timeout for manager
<b>mediation client section</b>		
medcli.database		Mediation client database URI
medcli.dpd	5m	DPD timeout to use in mediation client plugin
medcli.rekey	20m	Rekeying time on mediation connections in mediation client plugin
<b>mediation server section</b>		
medsrv.database		Mediation server database URI
medsrv.debug	no	Debugging in mediation server web application
medsrv.dpd	5m	DPD timeout to use in mediation server plugin
medsrv.load		Plugins to load in mediation server plugin
medsrv.password_length	6	Minimum password length required for mediation server user accounts
medsrv.rekey	20m	Rekeying time on mediation connections in mediation server plugin
medsrv.socket		Run Mediation server web application statically on socket
medsrv.threads	5	Number of thread for mediation service web application
medsrv.timeout	15m	Session timeout for mediation service
<b>openac section</b>		
openac.load		Plugins to load in ipsec openac tool
<b>pki section</b>		
pki.load		Plugins to load in ipsec pki tool
<b>pluto section</b>		
pluto.dns1		DNS server 1 assigned to peer via Mode Config, see <b>attr plugin</b>
pluto.dns2		DNS server 2 assigned to peer via Mode Config
pluto.load		Plugins to load in IKEv1 pluto daemon, also see <b>PluginLoad</b>
pluto.nbns1		WINS server 1 assigned to peer via Mode Config, see <b>attr plugin</b>
pluto.nbns2		WINS server 2 assigned to peer via Mode Config
pluto.threads	4	Number of worker threads in pluto
<b>pluto plugins section</b>		
pluto.plugins.attr		Section to specify arbitrary attributes that are assigned to a peer via Mode Config, see <b>attr plugin</b>

pluto.plugins.kernel-klips.ipsec_dev_count	4	Number of ipsecN devices
pluto.plugins.kernel-klips.ipsec_dev_mtu	0	Set MTU of ipsecN device
pool section		
pool.load		Plugins to load in ipsec pool tool
scepclient section		
scepclient.load		Plugins to load in ipsec scepclient tool
starter section		
starter.load_warning	yes	Disable charon/pluto plugin load option warning

strongswan.org

Wiki/Project Management

Downloads


Gitweb

主页

项目

帮助

登录 注册



strongSwan

搜索:

概述

活动

路线图

问题

Wiki

版本库

Developer Documentation


UserDocumentation »

历史记录

strongSwan plugins

The strongSwan distribution ships with a growing list of plugins. This allows us to add extended and specialized features, but keep the core as small as possible.

Many components of strongSwan come with a set of plugins. The plugins for libstrongswan provide cryptographic backends, URI fetchers and database layers. The plugins of libhydra are usable by the both IKE daemons, pluto and charon. libcharon comes with a large set of very specialized plugins for specific needs.

Plugin Name	E	S	Description
<i>E = Enabled by default</i> <i>S = Plugin status: s = stable, e = experimental, d = under development/incomplete</i>			
libstrongswan plugins			
aes	x	s	AES-128/192/256 cipher software implementation
af-alg		s	AF_ALG Linux crypto API interface, provides ciphers/hashers/hmac/xcbc
agent		s	RSA private key backend connecting to SSH-Agent
blowfish		s	Blowfish cipher software implementation
ccm		s	CCM cipher mode wrapper
constraints	x	s	X.509 certificate advanced constraint checking
ctr		s	CTR cipher mode wrapper
curl		s	libcurl based HTTP/FTP fetcher
des	x	s	DES/3DES cipher software implementation
dnskey	x	s	Parse  <b>RFC 4034</b> public keys
fips-prf	x	s	PRF specified by FIPS, used by EAP-SIM/AKA algorithms
gcm		s	GCM cipher mode wrapper
gcrypt		s	Crypto backend based on libgcrypt, provides RSA/DH/ciphers/hashers/rng
gmp	x	s	RSA/DH crypto backend based on libgmp
hmac	x	s	HMAC wrapper using various hashers
ldap		s	LDAP fetching plugin based on libldap
md4		s	MD4 hasher software implementation
md5	x	s	MD5 hasher software implementation
mysql		s	MySQL database backend based on libmysqlclient
openssl		s	Crypto backend based on OpenSSL, provides RSA/ECDSA/DH/ECDH/ciphers/hashers/X.509/CRL
padlock		e	VIA padlock crypto backend, provides AES128/SHA1
pem	x	s	PEM encoding/decoding routines
pgp	x	s	PGP encoding/decoding routines
pkcs11		s	PKCS#11 smartcard backend
pkcs1	x	s	PKCS#1 encoding/decoding routines
pubkey	x	s	Wrapper to handle raw public keys as trusted certificates
random	x	s	RNG reading from /dev/[u]random
revocation	x	s	X.509 CRL/OCSP revocation checking
sha1	x	s	SHA1 hasher software implementation
sha2	x	s	SHA256/SHA384/SHA512 hasher software implementation
soup		s	libsoup based HTTP fetcher
sqlite		s	SQLite database backend based on libsqlite3
test-vectors		s	Set of test vectors for various algorithms
x509	x	s	Advanced X.509 plugin for parsing/generating X.509 certificates/CRLs and OCSP messages

Documentation

IPsec Documentation

Installation Documentation

User Documentation

Developer Documentation

Resources

Changelog

Download

Gitweb

Wiki

起始页

按标题索引

按日期索引

http://wiki.strongswan.org/projects/strongswan/wiki/PluginList[2011-8-31 22:07:32]

xcbc	x	s	XCBC wrapper using various ciphers
libhydra plugins			
attr	x	s	Provides IKE attributes configured in strongswan.conf
attr-sql		s	Provides IKE attributes read from a database to peers
kernel-klips		e	IPsec kernel interface to an older KLIPS version
kernel-netlink	x	s	IPsec/Networking kernel interface using Linux Netlink
kernel-pfkey		e	IPsec kernel interface using PF_KEY
kernel-pfroute		e	Networking kernel interface using PF_ROUTE
resolve	x	s	Write name servers received via IKE to a resolv.conf file
pluto plugins			
xauth	x	s	XAUTH authentication
libcharon plugins			
addrblock		s	Narrow traffic selectors to  <b>RFC 3779</b> address blocks in X.509 certificates
android		s	<b>Android</b> configuration/control backend, works with <b>Android strongSwan applet</b>
coupling		s	Permanent peer certificate coupling
certexpire		s	Export expiration dates of used certificates
dhcp		s	Forward <b>virtual IP</b> address pool lookup to a DHCP server
duplicheck		s	Advanced duplicate checking with liveness test and notifications
eap-aka		s	Generic EAP-AKA protocol handler using different backends
eap-aka-3gpp2		s	EAP-AKA backend implementing standard 3GPP2 algorithm in software
eap-gtc		s	EAP-GTC protocol handler authenticating against PAM
eap-identity		s	EAP-Identity identity exchange algorithm, to use with other EAP protocols
eap-md5		s	EAP-MD5 protocol handler using passwords
eap-mschapv2		s	EAP-MSCHAPv2 protocol handler using passwords/NT hashes
eap-peap		s	EAP-PEAP protocol handler, wraps other EAP methods securely
eap-radius		s	EAP server proxy plugin forwarding EAP conversations to a RADIUS server
eap-sim		s	Generic EAP-SIM protocol handler using different backends
eap-sim-file		s	EAP-SIM backend reading triplets from a file
eap-sim-pcsc		s	EAP-SIM backend based on a PC/SC smartcard reader
eap-simaka-pseudonym		s	EAP-SIM/AKA in-memory pseudonym identity database
eap-simaka-reauth		s	EAP-SIM/AKA in-memory reauthentication identity database
eap-simaka-sql		s	EAP-SIM/AKA backend reading triplets/quintuplets from a SQL database
eap-tls		s	EAP-TLS protocol handler, to authenticate with certificates in EAP
eap-tnc		s	EAP-TNC protocol handler, Trusted Network Connect in a TLS tunnel
eap-ttls		s	EAP-TTLS protocol handler, wraps other EAP methods securely
farp		s	Fakes ARP responses for requests to a <b>virtual IP address</b> assigned to a peer
ha		s	High-Availability clustering
led		s	Let Linux LED subsystem LEDs blink on IKE activity
load-tester		s	Perform IKE load tests against self or a gateway
maemo		e	Maemo 5 configuration/control backend, works with Maemo strongSwan applet
medcli		d	Web interface based mediation client interface
medsrv		d	Web interface based mediation server interface
nm		s	NetworkManager configuration/control backend, works with NetworkManager strongSwan applet
smp		d	XML based strongSwan Management Protocol
socket-default	*	s	Default socket implementation for IKE messages, enabled if pluto disabled
socket-dynamic		e	Dynamic binding socket implementation, capable of sending IKE messages on any port
socket-raw	*	s	RAW socket allowing charon to run parallel with pluto, enabled if pluto enabled
sql		s	SQL configuration backend reading configurations/credentials from a database
stroke	x	s	Stroke configuration/control backend, to use with ipsec script and starter

<b>tnccs-11</b>		s	Trusted Network Connect protocol version 1.1
<b>tnccs-20</b>		s	Trusted Network Connect protocol version 2.0
<b>tnccs-dynamic</b>		s	Trusted Network Connect Dynamic protocol discovery
<b>tnc-ifmap</b>		s	Trusted Network Connect IF-MAP 2.0 client
<b>tnc-imc</b>		s	Trusted Network Connect Integrity Measurement Collectors
<b>tnc-imv</b>		s	Trusted Network Connect Integrity Measurement Validators
uci		d	OpenWRT UCI configuration backend
unit-tests		d	Unit tests to run during daemon startup
updown	x	s	Shell script invocation during tunnel up/down events
<b>whitelist</b>		s	Check authenticated identities against a whitelist

strongswan.org

Wiki/Project Management

Downloads


Gitweb

主页

项目

帮助

登录 注册



strongSwan

搜索:

概述

活动

路线图

问题

Wiki


版本库

Developer Documentation


strongSwan Developer Documentation

历史记录



Getting the Source Code

The easiest way to get the source code is checking it out from our  **Git** repository:



```
git clone git://git.strongswan.org/strongswan.git
```

The *master* branch is also mirrored to  **GitHub**.

Browsing the Source Code

The Git repository can be browsed online using either  **Gitweb** or the integrated  **repository browser** (slower).

Source Code Documentation

The newer parts of strongSwan use extractable inline documentation extensively. This documentation is extracted with **doxygen** for the latest release and uploaded to  **strongswan.org/apidoc**, but can also be browsed here using the  **Developer Documentation** button above.

Code style

For new code (charon, libstrongswan, ...) we heavily use an **object oriented programming style** for C. Also have a look to our basic **programming style** guidelines.

Contributions

Before starting development, please read our **contribution requirements**.

Components

The *src* directory in the strongSwan distribution contains a set of compoments:

<b>charon</b>	The IKEv2 keying daemon.
<b>pluto</b>	The IKEv1 keying daemon.
<b>libstrongswan</b>	The strongSwan library with basic functions used by the daemons and utilities
<b>libcharon</b>	Contains most of the code and the plugins of the charon daemon
<b>libhydra</b>	Contains daemon-specific code and plugins used by both the charon und pluto daemons
libfreeswan	The deprecated library used by pluto.
<b>dumm</b>	An new and experimental UML testing framework in development.
<b>ipsec</b>	The ipsec command line tool wrapping commands and other tools.
<b>libfast</b>	A lightweight framework to build native web applications using ClearSilver and FastCGI.
<b>manager</b>	A graphical management application for charon based on libfast.
<b>openac</b>	Utility to build attribute certificates on the command line.
<b>pki</b>	Public Key Infrastructure utility.
<b>scepclient</b>	Utility to enroll certificates using the SCEP protocol.
<b>starter</b>	Reads <i>ipsec.conf</i> and controls the keying daemons pluto and charon.
<b>stroke</b>	Command line utility to control charon via the stroke protocol.
<b>whack</b>	Command line utility to control pluto via the whack protocol.

Documentation

**IPsec Documentation**

**Installation Documentation**

**User Documentation**

**Developer Documentation**

Resources

**Changelog**

**Download**

**Gitweb**

Wiki

起始页

按标题索引

按日期索引

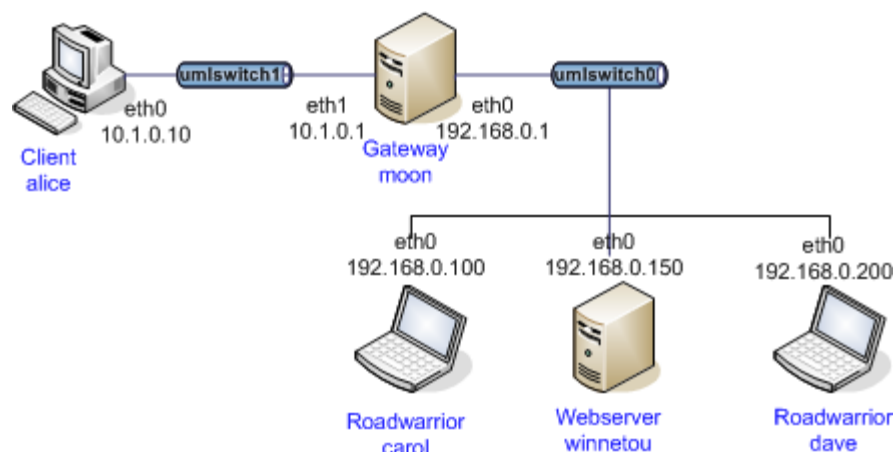
Powered by **Redmine** © 2006-2011 Jean-Philippe Lang

# Test ike/rw-cert

## Description

Roadwarrior **carol** sets up an IKEv1 connection and roadwarrior **dave** an IKEv2 tunnel, respectively, to the gateway **moon**. In order to test the established tunnels, both roadwarriors ping the client **alice** in the subnet behind gateway **moon**.

- [console.log](#)



## moon

- [ipsec.conf](#)
- [ipsec.secrets](#)
- [ipsec.sql](#)
- [strongswan.conf](#)
- [ipsec.statusall](#)
- [ipsec.listall](#)
- [auth.log](#)
- [daemon.log](#)
- [ip -s xfrm policy](#)
- [ip -s xfrm state](#)
- [ip route list table 220](#)
- [iptables -L](#)

## carol

- [ipsec.conf](#)
- [ipsec.secrets](#)
- [ipsec.sql](#)
- [strongswan.conf](#)
- [ipsec.statusall](#)
- [ipsec.listall](#)
- [auth.log](#)
- [daemon.log](#)
- [ip -s xfrm policy](#)
- [ip -s xfrm state](#)
- [ip route list table 220](#)
- [iptables -L](#)

## dave

- [ipsec.conf](#)
- [ipsec.secrets](#)
- [ipsec.sql](#)
- [strongswan.conf](#)
- [ipsec.statusall](#)
- [ipsec.listall](#)
- [auth.log](#)
- [daemon.log](#)
- [ip -s xfrm policy](#)
- [ip -s xfrm state](#)
- [ip route list table 220](#)
- [iptables -L](#)



[Login](#)

[ESP\\_Preferences](#)

- [FrontPage](#)
- [RecentChanges](#)
- [FindPage](#)
- [HelpContents](#)
- [ESP\\_Preferences](#)
- Immutable Page
- [Info](#)
- [Attachments](#)
- More Actions:

## ESP Payload Decryption / ESP Authentication Checking

If Libgcrypt is linked with Wireshark you can decrypt ESP Payloads and/or Authentication Checking. You can see if your version of Wireshark supports ESP decryption by looking for "with Gcrypt" in the about box.



The following encryption algorithms are supported:

- NULL Encryption.
- TripleDES-CBC [RFC2451](#) with key length of 192 bits.
- AES-CBC with 128-bit keys [RFC3602](#) with key length of 128/192/256 bits.
- AES-CTR [RFC3686](#) with key length of 160/224/288 bits. The remaining 32 bits will be used as nonce.
- DES-CBC [RFC2405](#) with key length of 64 bits.
- BLOWFISH-CBC [RFC2451](#) with key length of only 128 bits.
- TWOFISH-CBC with key of 128/256 bits.

The following authentication algorithms are supported:

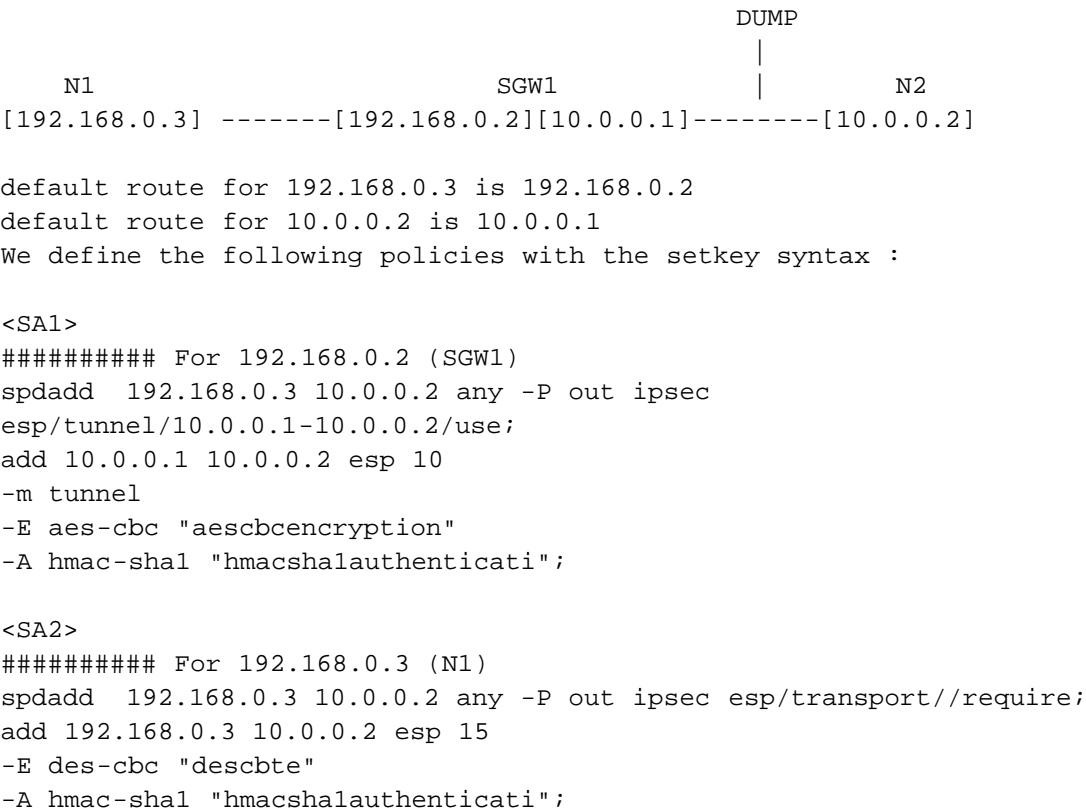
- NULL Authentication.
- HMAC-SHA1-96 [RFC2404](#) : any keylen
- HMAC-MD5-96 [RFC2403](#) : any keylen
- HMAC-SHA256 : any keylen

## IPsec Modes

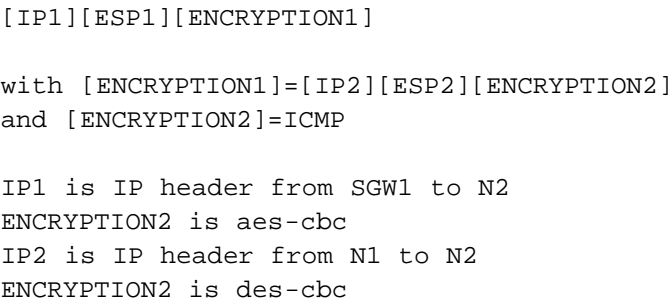


IPsec may be used in two Modes : tunnel or transport and concerns two kinds of nodes : End Nodes and Secure Gateways. Each kind of node may use IPsec using these two Modes. This dissector aim is to decrypt the whole packet if you have enough information concerning the different Security Associations.

Here is one of the more complex topology (if you have ESP in tunnel Mode in ESP in tunnel Mode ... it should work the same).



It means that packets coming from N1 to N2 will be encrypted with des-cbc and tunneled from SGW1 with ESP encryption aes-cbc to N2. If we have a look at the DUMP host, we have only two SAs to decrypt the entire packet. If we have a look at the different Layers it will be :



Thus, the IPsec dissector knowing these two SAs, will decrypt first ENCRYPTION1 using SA1, will dissect it, will get ENCRYPTION2, will decrypt it using SA2 and will dissect it getting the full decrypted packet.

If you look at the Authentication field, you will notice that 2 fields are available. The inner one is available only if ENCRYPTION1 is decrypted.

Here's what a packet looks like after decryption:



## ESP Preferences

In order to decrypt ESP Payload or to check ESP Authenticator, you need to give some elements of the known Security Associations (SA). I.e :

- The Source Address of the SA. Either IPv6, either IPv4.
- The Destination Address of the SA. Either IPv6, either IPv4.
- The SPI (Security Parameter Index).
- The Encryption algorithm and the associated key.
- The Authentication algorithm; even if you do not want to check Authentication, it has some impact on the bytes alignment.

We do not need to indicate the operation Mode (transport, tunnel), the decryption will be done iteratively.

These different elements have to be indicated in The ESP Preferences Menu of Wireshark. Indeed, it would perhaps have been better to set it in a separate file, but it is quite convenient to have at least a few rules in the Preferences Box. Moreover the parsing is quite basic, without the use of any library/tool such as Lex/yacc for example.

If you need to modify the number of Security Associations, you have to do it in the dissector by modifying the value of "IPSEC\_NB\_SA".

Here are the preferences used to decrypt the previous screen:



### Attempt to detect/decode NULL encrypted ESP payloads

This field is an heuristic in order to decrypt packet. It assumes that packet are encrypted using the NULL algorithm and the Authentication uses 12 bytes as with hmac-sha1-96/hmac-sha256/hmac-md5-96. This field Should be the only one available if Wireshark is not linked with libgcrypt. Otherwise, if set, all packets that are not caught by the Security Associations are decrypted using this heuristic. No authentication Checking will be done on these packets.

### Attempt to detect/decode encrypted ESP payloads

This field Should be available only if Wireshark is linked with libgcrypt. It is used to activate the Security Associations. When an IPsec ESP packet will be caught by a Security Association (Source/Destination/SPI) it will be decrypted using the specified Encryption/Authentication Algorithm and the associated Encryption Key. This checking will be done iteratively.

### Attempt to check ESP Authentication

This field Should be available only if Wireshark is linked with libgcrypt. When an IPsec ESP packet will be caught by a Security Association (Source/Destination/SPI) the Authentication will be checked using the specified Authentication Algorithm and the associated Authentication Key. This checking will be done iteratively.

### Security Associations And SA Filters

This field uses the following syntax (with spaces or not):

Protocol | Source Address | Destination Address | SPI

Where :

- Protocol: either IPv4, IPv6 (upper and/or lowercase letters)
- SPI : the Security Parameter Index of the Security Association. You may indicate it in decimal (ex: 123) or in hexadecimal (ex: 0x45). The special keywords '\*' may be used to match any SPI. Nevertheless, if you use more than one '\*', it will restrict the length of the SPI in decimal to as many '\*' as indicated. For example '\*\*\*' will match 23 but not 234. 234 will be match by '\*\*\*\*'. No checking will be done on the SPI value. Thus you have to take into account that the SPI is 4 bytes length.
- Addresses : In this field we may have IPv6 or IPv4 address. Any address is a combination of an address or a prefix and a Prefixlen/Netmask separated by '/'. You may omit the Prefixlen/Netmask, assuming that the Address is 128 bits length for IPv6 and 32 bits length for IPv4. The character '\*' used at the Prefixlen/Netmask position will be as if you had omit it.
  - IPv6 Addresses : Any valid IPv6 address is accepted. ex: 3FFE::1/128, 3FFE:4:5:6666::/64, ::1/128, 3FFE:4::5. If your address is incorrect and longer than 16 bytes, only the last 16 bytes will be taken into account. You also may use the special character '\*' to indicate any 4 bits block. ie : 3ffe::45\*6. If you use only one '\*' in the Address field it will accept any IPv6 address.
  - IPv4 Addresses : Any valid IPv4 address is accepted. ex : 190.0.0.1/24, 10.0.0.2. You also may use the special character '\*' to indicate any 8 bits block. ie 190.\*.\*.3. If you use only one '\*' in the Address field it will accept any IPv4 address. No checking of correct IPv4 address will be done. For example 456.345.567.890 will be accepted. Thus you have to take care about what you write. Nevertheless only 3 characters will be taken into account for one byte. Ex : 190.0.0.0184 will not be considered correct. (Instead a kind of LRU Mechanism will be used and the address taken into account will be 190.0.0.418). Moreover only the four first values will be used (ie 190.0.0.12.13 will be considered as 190.0.0.12).

## Encryption, Authentication Algorithms & Keys

In these fields you have to describe the Authentication, Encryption Algorithms and the Authentication, Encryption Keys for the SAs. The key sizes should be conformant with what is specified in the Algorithms otherwise it will not work. Keys may be written in Decimal or in Hexadecimal (beginning with 0x).

We may notice that if we only care about decryption, choosing one of hmac-sha1-96/aes-xcbc-mac-96/hmac-md5-96 for Authentication will have no impact on the decryption since all this algorithm will give a 12-bytes authenticator field. Thus either we choose one of it without the "FIELD: Attempt to check ESP Authentication" set or we may use the Authentication algorithm "Any 12 Byte Of authentication [No checking]". In this case, the "FIELD: Attempt to check ESP Authentication" has no impact on the decision. The algorithms explicitly taken into account are the followings :

- TripleDES-CBC [RFC2451](#) :

According to RFC 2451, 3DES CBC uses a key of 192 bits. The first 3DES key is taken from the first 64 bits, the second from the next 64 bits, and the third from the last 64 bits. Implementations MUST take into consideration the parity bits when initially accepting a new set of keys. Each of the three keys is really 56 bits in length with the extra 8 bits used for parity. 3DES CBC uses an IV of 8 octets.

- AES-CBC with 128-bit keys [RFC3602](#) :

According to RFC 3602, AES supports three key sizes: 128 bits, 192 bits, and 256 bits. The default key size is 128 bits, and all implementations MUST support this key size. Implementations MAY also support key sizes of 192 bits and 256 bits. AES-CBC uses an IV of 16 octets.

- AES-CTR [RFC3686](#) :

According to RFC 3686, AES supports three key sizes: 128 bits, 192 bits, and 256 bits. The default key size is 128 bits, and all implementations MUST support this key size. Implementations MAY also support key sizes of 192 bits and 256 bits. AES-CTR uses an IV of 8 octets.

- DES-CBC [RFC2405](#) :

According to RFC 2405, DES-CBC is a symmetric secret key algorithm. The key size is 64-bits. It is commonly known as a 56-bit key as the key has 56 significant bits; the least significant bit in every byte is the parity bit. DES-CBC uses an IV of 8 octets.

- BLOWFISH-CBC [RFC2451](#) :

Bruce Schneier of Counterpane Systems developed the Blowfish cipher algorithm. RFC 2451 shows that Blowfish uses key sizes from 40 to 448 bits. The Default size is 128 bits. We will only accept key sizes of 128 bits, because libgrypt only accept this key size. Have a look to <http://www.schneier.com> for more information. BLOWFISH-CBC uses an IV of 8 octets.

- TWOFISH-CBC :

Twofish is a 128-bit block cipher developed by Counterpane Labs that accepts a variable-length key up to 256 bits. We will only accept key sizes of 128 and 256 bits. Have a look to <http://www.schneier.com> for more information. TWOFISH-CBC uses an IV of 16 octets.

- HMAC-MD5-96 [RFC2403](#) :

HMAC with MD5 provides data origin Authentication and integrity protection. HMAC-MD5-96 produces a 128-bit authenticator value. For use with either ESP or AH, a truncated value using the first 96 bits MUST be supported. Upon sending, the truncated value is stored within the authenticator field. Upon receipt, the entire 128-bit value is computed and the first 96 bits are compared to the value stored in the authenticator field. No other authenticator value lengths are supported by HMAC-MD5-96.

- HMAC-SHA1-96 [RFC2404](#) :

SHA-1 combined with HMAC [RFC2104] provides a keyed Authentication mechanism. HMAC-SHA-1-96 produces a 160-bit authenticator value. For use with either ESP or AH, a truncated value using the first 96 bits MUST be supported. Upon sending, the truncated value is stored within the authenticator field. Upon receipt, the entire 160-bit value is computed and the first 96 bits are compared to the value stored in the authenticator field. No other authenticator value lengths are supported by HMAC-SHA-1-96.

- HMAC-SHA256

This is the SHA-256 algorithm which yields a message digest of 32 bytes. For use with either ESP or AH, a truncated value using the first 96 bits MUST be supported. Upon sending, the truncated value is stored within the authenticator field. Upon receipt, the entire 128-bit value is computed and the first 96 bits are compared to the value stored in the authenticator field. Our implementation will support any key length.

## Possible Extensions

For sure, you may use some others ESP Encryption/Authentication algorithms. and it should not be very difficult to add some other ones. It also should be possible to adapt this to check AH Authenticator and why not to do things for IPComp. If the maximum number of Security Associations fixed is a problem for you, you may modify this value in the dissector "IPSEC\_NB\_SA". It could also be interesting to keep a few SAs in the Preferences Box and have a way to add some more in a separate file.

### [CategoryHowTo](#)

ESP\_Preferences (last edited 2010-03-02 01:48:38 by [GeraldCombs](#))

- Immutable Page
- [Info](#)
- [Attachments](#)
- More Actions:

Original content on this site is available under the GNU General Public License.  
See the [License](#) page for details.

Powered by [MoinMoin](#) and [Python](#).  
Please don't pee in the pool.