ipsec.conf(5) - Linux man page

Name

ipsec.conf - IPsec configuration and connections

Description

The optional *ipsec.conf* file specifies most configuration and control information for the Openswan IPsec subsystem. (The major exception is secrets for authentication; see **ipsec.secrets**(5).) Its contents are not security-sensitive *unless* manual keying is being done for more than just testing, in which case the encryption/authentication keys in the descriptions for the manually-keyed connections are very sensitive (and those connection descriptions are probably best kept in a separate file, via the include facility described below).

The file is a text file, consisting of one or more *sections*. White space followed by **#** followed by anything to the end of the line is a comment and is ignored, as are empty lines which are not within a section.

A line which contains **include** and a file name, separated by white space, is replaced by the contents of that file, preceded and followed by empty lines. 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. Only a single filename may be supplied, and it may not contain white space, but it may include shell wildcards (see **sh**(1)); for example:

include ipsec.*.conf

The intention of the include facility is mostly to permit keeping information on connections, or sets of connections, separate from the main configuration file. This permits such connection descriptions to be changed, copied to the other security gateways involved, etc., without having to constantly extract them from the configuration file and then insert them back into it. Note also the **also** and **alsoflip** parameters (described below) which permit splitting a single logical section (e.g. a connection description) into several actual sections.

The first significant line of the file must specify the version of this specification that it conforms to:

version 2

A section begins with a line of the form:

type name

where *type* indicates what type of section follows, and *name* is an arbitrary name which distinguishes the section from others of the same type. (Names must start with a letter and may contain only letters, digits, periods, underscores, and hyphens.) All subsequent non-empty lines which begin with white space are part of the section; comments within a section must begin with white space too. There may be only one section of a given type with a given name.

Lines within the section are generally of the form

*parameter***=***value*

(note the mandatory preceding white space). There can be white space on either side of the **=**. Parameter names follow the same syntax as section names, and are specific to a section type. Unless otherwise explicitly specified, no parameter name may appear more than once in a section.

An empty *value* stands for the system default value (if any) of the parameter, i.e. it is roughly equivalent to omitting the parameter line entirely. A *value* may contain white space only if the entire *value* is enclosed in double quotes (**"**); a *value* cannot itself contain a double quote, nor may it be continued across more than one line.

Numeric values are specified to be either an "integer" (a sequence of digits) or a "decimal number" (sequence of digits optionally followed by '.' and another sequence of digits).

There is currently one parameter which is available in any type of section:

also

the value is a section name; the parameters of that section are appended to this section, as if they had been written as part of it. The specified section must exist, must follow the current one, and must have the same section type. (Nesting is permitted, and there may be more than one **also** in a single section, although it is forbidden to append the same section more than once.) This allows, for example, keeping the encryption keys for a connection in a separate file from the rest of the description, by using both an **also** parameter and an **include** line. (Caution, see BUGS below for some restrictions.)

alsoflip

can be used in a **conn** section. It acts like an **also** that flips the referenced section's entries left-for-right.

Parameter names beginning with **x-** (or **X-**, or **x_**, or **X_**) are reserved for user extensions and will never be assigned meanings by IPsec. Parameters with such names must still observe the syntax rules (limits on characters used in the name; no white space in a non-quoted value; no newlines or double quotes within the value). All other as-yet-unused parameter names are reserved for future IPsec improvements.

A section with name **%default** specifies defaults for sections of the same type. For each parameter in it, any section of that type which does not have a parameter of the same name gets a copy of the one from the **%default** section. There may be multiple **%default** sections of a given type, but only one default may be supplied for any specific parameter name, and all **%default** sections of a given type must precede all non-**%default** sections of that type. **%default** sections may not contain **also** or **alsoflip** parameters.

Currently there are two types of section: a **config** section specifies general configuration information for IPsec, while a **conn** section specifies an IPsec connection.

Conn Sections

A **conn** section contains a *connection specification*, defining a network connection to be made using IPsec. The name given is arbitrary, and is used to identify the connection to **ipsec_auto**(8) and **ipsec_manual**(8). Here's a simple example:

```
conn snt
```

left=10.11.11.1

```
leftsubnet=10.0.1.0/24

leftnexthop=172.16.55.66

leftsourceip=10.0.1.1

right=192.168.22.1

rightsubnet=10.0.2.0/24

rightnexthop=172.16.88.99

rightsourceip=10.0.2.1

keyingtries=%forever
```

A note on terminology... In automatic keying, there are two kinds of communications going on: transmission of user IP packets, and gateway-to-gateway negotiations for keying, rekeying, and general control. The data path (a set of "IPsec SAs") used for user packets is herein referred to as the "connection"; the path used for negotiations (built with "ISAKMP SAs") is referred to as the "keying channel".

To avoid trivial editing of the configuration file to suit it to each system involved in a connection, connection specifications are written in terms of *left* and *right* participants, rather than in terms of local and remote. Which participant is considered *left* or *right* is arbitrary; IPsec figures out which one it is being run on based on internal information. This permits using identical connection specifications on both ends. There are cases where there is no symmetry; a good convention is to use *left* for the local side and *right* for the remote side (the first letters are a good mnemonic).

Many of the parameters relate to one participant or the other; only the ones for *left* are listed here, but every parameter whose name begins with **left** has a **right** counterpart, whose description is the same but with **left** and **right** reversed.

Parameters are optional unless marked "(required)"; a parameter required for manual keying need not be included for a connection which will use only automatic keying, and vice versa.

CONN PARAMETERS: GENERAL

The following parameters are relevant to both automatic and manual keying. Unless otherwise noted, for a connection to work, in general it is necessary for the two ends to agree exactly on the values of these parameters.

connaddrfamily

the connection addrress family of the connection; currently the accepted values are **ipv4** (the default); or **ipv6**,

The ipv6 family is currently only supported using the NETKEY stack.

type

the type of the connection; currently the accepted values are **tunnel** (the default) signifying a host-to-host, host-to-subnet, or subnet-to-subnet tunnel; **transport**, signifying host-to-host transport mode; **passthrough**, signifying that no IPsec processing should be done at all; **drop**, signifying that packets should be discarded; and **reject**, signifying that packets should be discarded and a diagnostic ICMP returned.

left

(required) the IP address of the left participant's public-network interface, in any form accepted by **ipsec_ttoaddr**(3). Currently, IPv4 and IPv6 IP addresses are supported. There are several magic values. If it is **%defaultroute**, and the **config setup** section's, **interfaces** specification contains **%defaultroute, left** will be filled in automatically with the local address of the default-route interface (as determined at IPsec startup time); this also overrides any value supplied for **leftnexthop**. (Either **left** or **right** may be **%defaultroute**, but not both.) The value **%any** signifies an address to be filled in (by automatic keying) during negotiation. The value **%opportunistic** signifies that both **left** and **leftnexthop** are to be filled in (by automatic keying) from DNS data for **left**'s client. The value can also contain the interface name, which will then later be used to obtain the IP address from to fill in. For example **%ppp0** The values **%group** and **%opportunisticgroup** makes this a policy group conn: one that will be instantiated into a regular or opportunistic conn for each CIDR block listed in the policy group file with the same name as the conn.

If using IP addresses in combination with NAT, always use the actual local machine's (NAT'ed) IP address, and if the remote (eg right=) is NAT'ed as well, the remote's public (**not** NAT'ed) IP address. Note that this makes the configuration no longer symmetrical on both sides, so you cannot use an identical configuration file on both hosts.

leftsubnet

private subnet behind the left participant, expressed as *network***/***netmask* (actually, any form acceptable to **ipsec_ttosubnet**(3)); Currentlly, IPv4 and IPv6 ranges are supported. if omitted, essentially assumed to be *left*/32, signifying that the left end of the connection goes to the left participant only

leftsubnets

specify multiple private subnets behind the left participant, expressed as { *networkA***/***netmaskA networkB***/***netmaskB [...]* } If both a leftsubnets= and rightsubnets= is defined, all combinations of subnet tunnels will be instantiated. You cannot use leftsubnet and leftsubnets together. For examples see *testing/pluto/multinet-**.

leftprotoport

allowed protocols and ports over connection, also called Port Selectors. The argument is in the form *protocol*, which can be a number or a name that will be looked up in */etc/protocols*, such as *leftprotoport=icmp*, or in the form of *protocol/port*, such as *tcp/smtp*. Ports can be defined as a number (eg. 25) or as a name (eg smtp) which will be looked up in */etc/services*. A special keyword *%any* can be used to allow all ports of a certain protocol. The most common use of this option is for L2TP connections to only allow l2tp packets (UDP port 1701), eg: *leftprotoport=17/1701*. Some clients, notably older Windows XP and some Mac OSX clients, use a random high port as source port. In those cases *rightprotoport=17/%any* can be used to allow all UDP traffic on the connection. Note that this option is part of the proposal, so it cannot be arbitrarily left out if one end does not care about the traffic selection over this connection - both peers have to agree. The Port Selectors show up in the output of *ipsec eroute* and *ipsec auto --status* eg:*"l2tp": 193.110.157.131[@aivd.xelernace.com]:7/1701...%any:17/1701* This option only filters outbound traffic. Inbound traffic selection must still be based on firewall rules activated by an updown script. The variablees $PLUTO_MY_PROTOCOL, $PLUTO_PEER_PROTOCOL, $PLUTO_MY_PORT, and $PLUTO_PEER_PORT are available for use in *updown* scripts. Older workarounds for bugs involved a setting of *17/0* to denote *any single UDP* *port* (not UDP port 0). Some clients, most notably OSX, uses a random high port, instead of port 1705 for L2TP.

leftnexthop

next-hop gateway IP address for the left participant's connection to the public network; defaults to **%direct** (meaning *right*). If the value is to be overridden by the **left=%defaultroute** method (see above), an explicit value must *not* be given. If that method is not being used, but **leftnexthop** is **%defaultroute**, and **interfaces=%defaultroute** is used in the **config setup** section, the next-hop gateway address of the default-route interface will be used. The magic value **%direct** signifies a value to be filled in (by automatic keying) with the peer's address. Relevant only locally, other end need not agree on it.

leftsourceip

the IP address for this host to use when transmitting a packet to the other side of this link. Relevant only locally, the other end need not agree. This option is used to make the gateway itself use its internal IP, which is part of the leftsubnet, to communicate to the rightsubnet or right. Otherwise, it will use its **nearest** IP address, which is its public IP address. This option is mostly used when defining subnet-subnet connections, so that the gateways can talk to each other and the subnet at the other end, without the need to build additional host-subnet, subnet-host and host-host tunnels. Both IPv4 and IPv6 addresses are supported.

leftupdown

what "updown" script to run to adjust routing and/or firewalling when the status of the connection changes (default **ipsec _updown**). May include positional parameters separated by white space (although this requires enclosing the whole string in quotes); including shell metacharacters is unwise. An example to enable routing when using the NETKEY stack, one can use:

```
leftupdown="ipsec _updown --route yes"
```

See **ipsec_pluto**(8) for details. Relevant only locally, other end need not agree on it.

leftfirewall

This option is obsolete and should not used anymore.

If one or both security gateways are doing forwarding firewalling (possibly including masquerading), and this is specified using the firewall parameters, tunnels established with IPsec are exempted from it so that packets can flow unchanged through the tunnels. (This means that all subnets connected in this manner must have distinct, non-overlapping subnet address blocks.) This is done by the default *updown* script (see **ipsec_pluto**(8)).

The implementation of this makes certain assumptions about firewall setup, and the availability of the *Linux Advanced Routing* tools. In situations calling for more control, it may be preferable for the user to supply his own *updown* script, which makes the appropriate adjustments for his system.

CONN PARAMETERS: AUTOMATIC KEYING

The following parameters are relevant only to automatic keying, and are ignored in manual keying. Unless otherwise noted, for a connection to work, in general it is necessary for the two ends to agree exactly on the values of these parameters.

auto

what operation, if any, should be done automatically at IPsec startup; currently-accepted values are **add** (signifying an **ipsec auto --add**), **route** (signifying that plus an **ipsec auto --route**),

start (signifying that plus an **ipsec auto --up**), **manual** (signifying an **ipsec manual --up**), and **ignore** (also the default) (signifying no automatic startup operation). See the **config setup** discussion below. Relevant only locally, other end need not agree on it (but in general, for an intended-to-be-permanent connection, both ends should use **auto=start** to ensure that any reboot causes immediate renegotiation).

authby

how the two security gateways should authenticate each other; acceptable values are **secret** for shared secrets, **rsasig** for RSA digital signatures (the default), **secret|rsasig** for either, and **never** if negotiation is never to be attempted or accepted (useful for shunt-only conns). Digital signatures are superior in every way to shared secrets.

ike

IKE encryption/authentication algorithm to be used for the connection (phase 1 aka ISAKMP SA). The format is *"cipher-hash;modpgroup, cipher-hash;modpgroup, ..."* Any left out option will be filled in with all allowed default options. Multiple proposals are separated by a comma. If an **ike=** line is specified, no other received proposals will be accepted. Formerly there was a distinction (by using a **"!"** symbol) between "strict mode" or not. That mode has been obsoleted. If an **ike=** option is specified, the mode is always strict, meaning no other received proposals will be accepted. Some examples are **ike=3des-sha1,aes-sha1**, **ike=aes**, **ike=aes128-md5;modp2048**, **ike=aes128-sha1;dh22**, **ike=3des-md5;modp1024,aes-sha1;modp1536** or **ike=modp1536**. The options must be suitable as a value of ipsec_spi(8)'s **--ike** option. The default is to use IKE, and to allow all combinations of:

```
cipher:                 3des or aes
hash:                   sha1 or md5
pfsgroup (DHgroup):     modp1024 or modp1536
```

If Openswan was compiled with extra INSECURE and BROKEN options, then the des (1des) and null cipher, as well as modp768 are available. This turns your VPN into a joke. Do not enable these options.

If openswan was compiled with USE_MODP_RFC5114 support, then Diffie-Hellman groups 22, 23 and 24 are also implemented as per RFC-5114. Instead of the modp key syntax, use the "dh" keyword, for example *ike=3des-sha1;dh23*

phase2

Sets the type of SA that will be produced. Valid options are: **esp** for encryption (the default), and **ah** for authentication only.

phase2alg

Specifies the algorithms that will be offered/accepted for a phase2 negotiation. If not specified, a secure set of defaults will be used. Sets are separated using comma's.

The default values are the same as for ike= Note also that not all ciphers available to the kernel (eg through CryptoAPI) are necessarilly supported here.

The format for ESP is ENC-AUTH followed by an optional PFSgroup. For instance, "3des-md5" or "aes256-sha1;modp2048" or "aes-sha1,aes-md5".

For RFC-5114 DH groups, use the "dh" keyword, eg "aes256-sha1;dh23"

The format for AH is AUTH followed by an optional PFSgroup. For instance, "md5" or "sha1;modp1536".

A special case is AES CCM, which uses the syntax of "phase2alg=aes_ccm_a-152-null"

sha2_truncbug

The default hash truncation for sha2_256 is 128 bits. Linux implemented the draft version which stated 96 bits. This option enables using the bad 96 bits version to interop with older linux kernels (unpatched version 2.6.33 and older) and openswan versions before 2.6.38. Currently the accepted values are **no**, (the default) signifying default IETF truncation of 128 bits, or **yes**, signifying 96 bits broken Linux kernel style truncation.

esp

This option is obsolete. Please use **phase2alg** instead.

ah

AH authentication algorithm to be used for the connection, e.g here. **hmac-md5** The options must be suitable as a value of **ipsec_spi**(8)'s **--ah** option. The default is not to use AH. If for some (invalid) reason you still think you need AH, please use esp with the null encryption cipher instead. Note also that not all ciphers available to the kernel (eg through CryptoAPI) are necessarilly supported here.

ikev2

IKEv2 (RFC4309) settings to be used. Currently the accepted values are **permit**, (the default) signifying no IKEv2 should be transmitted, but will be accepted if the other ends initiates to us with IKEv2; **never** or **no** signifying no IKEv2 negotiation should be transmitted or accepted; **propose** or **yes** signifying that we permit IKEv2, and also use it as the default to initiate; **insist**, signifying we only accept and receive IKEv2 - IKEv1 negotiations will be rejected.

If the ikev2= setting is set to **permit** or **propose**, Openswan will try and detect a "bid down" attack from IKEv2 to IKEv1. Since there is no standard for transmitting the IKEv2 capability with IKEv1, Openswan uses a special Vendor ID "CAN-IKEv2". If a fall back from IKEv2 to IKEv1 was detected, and the IKEv1 negotiation contains Vendor ID "CAN-IKEv2", Openswan will immediately attempt and IKEv2 rekey and refuse to use the IKEv1 connection. With an ikev2= setting of **insist**, no IKEv1 negotiation is allowed, and no bid down attack is possible.

sareftrack

Set the method of tracking reply packets with SArefs when using an SAref compatible stack. Currently only the *mast* stack supports this. Acceptable values are **yes** (the default), **no** or **conntrack**. This option is ignored when SArefs are not supported. This option is passed as PLUTO_SAREF_TRACKING to the *updown* script which makes the actual decisions whether to perform any iptables/ip_conntrack manipulation. A value of yes means that an IPSEC mangle table will be created. This table will be used to match reply packets. A value of conntrack means that additionally, subsequent packets using this connection will be marked as well, reducing the lookups needed to find the proper SAref by using the ip_conntrack state. A value of no means no IPSEC mangle table is created, and SAref tracking is left to a third-party (kernel) module. In case of a third party module, the SArefs can be relayed using the HAVE_STATSD deamon.

leftid

how the left participant should be identified for authentication; defaults to **left**. Can be an IP address (in any **ipsec_ttoaddr**(3) syntax) or a fully-qualified domain name preceded by **@** (which is used as a literal string and not resolved). The magic value **%fromcert** causes the ID to be set to a DN taken from a certificate that is loaded. Prior to 2.5.16, this was the default if a certificate was specified. The magic value **%none** sets the ID to no ID. This is included for completeness, as the ID may have been set in the default conn, and one wishes for it to default instead of being explicitly set. The magic value **%myid** stands for the current setting of *myid*. This is set in **config setup** or by **ipsec_whack**(8)), or, if not set, it is the IP address in

%defaultroute (if that is supported by a TXT record in its reverse domain), or otherwise it is the system's hostname (if that is supported by a TXT record in its forward domain), or otherwise it is undefined.

leftrsasigkey

the left participant's public key for RSA signature authentication, in RFC 2537 format using **ipsec_ttodata**(3) encoding. The magic value **%none** means the same as not specifying a value (useful to override a default). The value **%dnsondemand** (the default) means the key is to be fetched from DNS at the time it is needed. The value **%dnsonload** means the key is to be fetched from DNS at the time the connection description is read from *ipsec.conf*; currently this will be treated as **%none** if **right=%any** or **right=%opportunistic**. The value **%dns** is currently treated as **%dnsonload** but will change to **%dnsondemand** in the future. The identity used for the left participant must be a specific host, not **%any** or another magic value. The value **%cert** will load the information required from a certificate defined in **%leftcert** and automatically define leftid for you. **Caution:** if two connection descriptions specify different public keys for the same **leftid**, confusion and madness will ensue.

leftrsasigkey2

if present, a second public key. Either key can authenticate the signature, allowing for key rollover.

leftcert

If you are using **leftrsasigkey=%cert** this defines the certificate you would like to use. It should point to a X.509 encoded certificate file. If you do not specify a full pathname, by default it will look in /etc/ipsec.d/certs. If openswan has been compiled with **USE_LIBNSS=true**, then openswan will also check the NSS database for RSA keys. These can be software or hardware.

leftca

specifies the authorized Certificate Authority (CA) that signed the certificate of the peer. If undefined, it defaults to the CA that signed the certificate specified in **leftcert**. The special **rightca=%same** is implied when not specifying a **rightca** and means that only peers with certificates signed by the same CA as the leftca will be allowed. This option is only useful in complex multi CA certificate situations. When using a single CA, it can be safely omitted for both left and right.

leftsendcert

This option configures when Openswan will send X.509 certificates to the remote host. Acceptable values are **yes|always** (signifying that we should always send a certificate), **ifasked** (signifying that we should send a certificate if the remote end asks for it), and **no|never** (signifying that we will never send a X.509 certificate). The default for this option is **ifasked** which may break compatibility with other vendor's IPSec implementations, such as Cisco and SafeNet. If you find that you are getting errors about no ID/Key found, you likely need to set this to **always**. This per-conn option replaces the obsolete global **nocrsend** option.

leftxauthserver

Left is an XAUTH server. This can use PAM for authentication or md5 passwords in */etc/ipsec.d/passwd*. These are additional credentials to verify the user identity, and should not be confused with the XAUTH **group secret**, which is just a regular PSK defined in *ipsec.secrets*. The other side of the connection should be configured as **rightxauthclient**. XAUTH connections cannot rekey, so **rekey=no** should be specified in this conn. For further details on how to compile and use XAUTH, see README.XAUTH. Acceptable values are **yes** or **no** (the default).

leftxauthclient

Left is an XAUTH client. The xauth connection will have to be started interactively and cannot be configured using **auto=start**. Instead, it has to be started from the commandline using *ipsec auto --up connname*. You will then be prompted for the username and password. To setup an

XAUTH connection non-interactively, which defeats the whole purpose of XAUTH, but is regularly requested by users, it is possible to use a whack command - *ipsec whack --name baduser --ipsecgroup-xauth --xauthname badusername --xauthpass password --initiate* The other side of the connection should be configured as **rightxauthserver**. Acceptable values are **yes** or **no** (the default).

leftxauthusername
The XAUTH username associated with this XAUTH connection. The XAUTH password can be configured in the *ipsec.secrets* file.

leftmodecfgserver
Left is a Mode Config server. It can push network configuration to the client. Acceptable values are **yes** or **no** (the default).

leftmodecfgclient
Left is a Mode Config client. It can receive network configuration from the server. Acceptable values are **yes** or **no** (the default).

modecfgpull
Pull the Mode Config network information from the server. Acceptable values are **yes** or **no** (the default).

modecfgdns1, **modecfgdns2**, **modecfgwins1**, **modecfgwins2**
Specify the IP address for DNS or WINS servers for the client to use.

remote_peer_type
Set the remote peer type. This can enable additional processing during the IKE negotiation. Acceptable values are **cisco** or **ietf** (the default). When set to cisco, support for Cisco IPsec gateway redirection and Cisco obtained DNS and domainname are enabled. This includes automatically updating (and restoring) /etc/resolv.conf. These options require that XAUTH is also enabled on this connection.

nm_configured
Mark this connection as controlled by Network Manager. Acceptable values are **yes** or **no** (the default). Currently, setting this to yes will cause openswan to skip reconfiguring resolv.conf when used with XAUTH and ModeConfig.

forceencaps
In some cases, for example when ESP packets are filtered or when a broken IPsec peer does not properly recognise NAT, it can be useful to force RFC-3948 encapsulation. **forceencaps=yes** forces the NAT detection code to lie and tell the remote peer that RFC-3948 encapsulation (ESP in UDP port 4500 packets) is required. For this option to have any effect, the setup section option **nat_traversal=yes** needs to be set. Acceptable values are **yes** or **no** (the default).

overlapip
a boolean (yes/no) that determines, when *subnet=vhost: is used, if the virtual IP claimed by this states created from this connection can with states created from other connections.

Note that connection instances created by the Opportunistic Encryption or PKIX (x.509) instantiation system are distinct internally. They will inherit this policy bit.

The default is no.

This feature is only available with kernel drivers that support SAs to overlapping conns. At present only the (klips)mast protocol stack supports this feature.

dpddelay
Set the delay (in seconds) between Dead Peer Dectection (RFC 3706) keepalives (R_U_THERE, R_U_THERE_ACK) that are sent for this connection (default 30 seconds). If dpddelay is set,

dpdtimeout also needs to be set.

dpdtimeout

Set the length of time (in seconds) we will idle without hearing either an R_U_THERE poll from our peer, or an R_U_THERE_ACK reply. After this period has elapsed with no response and no traffic, we will declare the peer dead, and remove the SA (default 120 seconds). If dpdtimeout is set, dpdaction also needs to be set.

dpdaction

When a DPD enabled peer is declared dead, what action should be taken. **hold** (default) means the eroute will be put into %hold status, while **clear** means the eroute and SA with both be cleared. **restart** means the the SA will immediately be renegotiated, and **restart_by_peer** means that *ALL* SA's to the dead peer will renegotiated.

dpdaction=clear is really only useful on the server of a Road Warrior config.

pfs

whether Perfect Forward Secrecy of keys is desired on the connection's keying channel (with PFS, penetration of the key-exchange protocol does not compromise keys negotiated earlier); Since there is no reason to ever refuse PFS, Openswan will allow a connection defined with **pfs=no** to use PFS anyway. Acceptable values are **yes** (the default) and **no**.

pfsgroup

This option is obsoleted, please use phase2alg if you need the pfs to be different from phase1 (the default) using: phase2alg=aes128-md5;modp1024

aggrmode

Use Aggressive Mode instead of Main Mode. Aggressive Mode is less secure, and vulnerable to Denial Of Service attacks. It is also vulnerable to brute force attacks with software such as **ikecrack**. It should not be used, and it should especially not be used with XAUTH and group secrets (PSK). If the remote system administrator insists on staying irresponsible, enable this option.

Aggressive Mode is further limited to only proposals with one DH group as there is no room to negotiate the DH group. Therefor it is mandatory for Aggressive Mode connections that both **ike=** and **phase2alg=** options are specified with only fully specified proposal using one DH group. Acceptable values are **yes** or **no** (the default).

The ISAKMP SA is created in exchange 1 in aggressive mode. Openswan has to send the exponent during that exchange, so it has to know what DH group to use before starting. This is why you can not have multiple DH groups in aggressive mode. In IKEv2, which uses a similar method to IKEv1 Aggressive Mode, there is a message to convey the DH group is wrong, and so an IKEv2 connection can actually recover from picking the wrong DH group by restarting its negotiation.

salifetime

how long a particular instance of a connection (a set of encryption/authentication keys for user packets) should last, from successful negotiation to expiry; acceptable values are an integer optionally followed by **s** (a time in seconds) or a decimal number followed by **m**, **h**, or **d** (a time in minutes, hours, or days respectively) (default **8h**, maximum **24h**). Normally, the connection is renegotiated (via the keying channel) before it expires. The two ends need not exactly agree on **salifetime**, although if they do not, there will be some clutter of superseded connections on the end which thinks the lifetime is longer.

The keywords "keylife" and "lifetime" are aliases for "salifetime."

rekey

whether a connection should be renegotiated when it is about to expire; acceptable values are **yes** (the default) and **no**. The two ends need not agree, but while a value of **no** prevents Pluto from requesting renegotiation, it does not prevent responding to renegotiation requested from the other end, so **no** will be largely ineffective unless both ends agree on it.

rekeymargin

how long before connection expiry or keying-channel expiry should attempts to negotiate a replacement begin; acceptable values as for **salifetime** (default **9m**). Relevant only locally, other end need not agree on it.

rekeyfuzz

maximum percentage by which **rekeymargin** should be randomly increased to randomize rekeying intervals (important for hosts with many connections); acceptable values are an integer, which may exceed 100, followed by a '%' (default set by **ipsec_pluto**(8), currently **100%**). The value of **rekeymargin**, after this random increase, must not exceed **salifetime**. The value **0%** will suppress time randomization. Relevant only locally, other end need not agree on it.

keyingtries

how many attempts (a whole number or **%forever**) should be made to negotiate a connection, or a replacement for one, before giving up (default **%forever**). The value **%forever** means "never give up" (obsolete: this can be written 0). Relevant only locally, other end need not agree on it.

ikelifetime

how long the keying channel of a connection (buzzphrase: "ISAKMP SA") should last before being renegotiated; acceptable values as for **keylife** (default set by **ipsec_pluto**(8), currently **1h**, maximum **24h**). The two-ends-disagree case is similar to that of **keylife**.

compress

whether IPComp compression of content is proposed on the connection (link-level compression does not work on encrypted data, so to be effective, compression must be done *before* encryption); acceptable values are **yes** and **no** (the default). The two ends need not agree. A value of **yes** causes IPsec to propose both compressed and uncompressed, and prefer compressed. A value of **no** prevents IPsec from proposing compression; a proposal to compress will still be accepted.

metric

Set the metric for the routes to the ipsecX or mastX interface. This makes it possible to do host failover from another interface to ipsec using route management. This value is passed to the _updown scripts as PLUTO_METRIC. This option is only available with KLIPS or MAST on Linux. Acceptable values are positive numbers, with the default being **1**.

disablearrivalcheck

whether KLIPS's normal tunnel-exit check (that a packet emerging from a tunnel has plausible addresses in its header) should be disabled; acceptable values are **yes** and **no** (the default). Tunnel-exit checks improve security and do not break any normal configuration. Relevant only locally, other end need not agree on it.

failureshunt

what to do with packets when negotiation fails. The default is **none**: no shunt; **passthrough**, **drop**, and **reject** have the obvious meanings.

CONN PARAMETERS: MANUAL KEYING

This command was obsoleted around the same time that Al Gore invented the internet. ipsec manual was used in the jurassic period to load static keys into the kernel. There are no rational reasons to use this, and it is not supported anymore. If you need to create static SAs, then you can use **ipsec spi** and **ipsec eroute** when using KLIPS or **ip xfrm** or **setkey** when using NETKEY.

No rational person uses static keys. They are not easier to use. REPEAT: they are not easier to use.

Config Sections

At present, the only **config** section known to the IPsec software is the one named **setup**, which contains information used when the software is being started (see **ipsec_setup**(8)). Here's an example:

```
config setup
```

interfaces="ipsec0=eth1 ipsec1=ppp0"

klipsdebug=none

plutodebug=control

protostack=auto

manualstart=

Parameters are optional unless marked "(required)".

The currently-accepted *parameter* names in a **config setup** section are:

myid

the identity to be used for **%myid**. **%myid** is used in the implicit policy group conns and can be used as an identity in explicit conns. If unspecified, **%myid** is set to the IP address in **%defaultroute** (if that is supported by a TXT record in its reverse domain), or otherwise the system's hostname (if that is supported by a TXT record in its forward domain), or otherwise it is undefined. An explicit value generally starts with ''**@**''.

protostack
decide which protocol stack is going to be used. Valid values are "auto", "klips", "netkey" and "mast". The "mast" stack is a variation for the klips stack.

interfaces
virtual and physical interfaces for IPsec to use: a single *virtual*=*physical* pair, a (quoted!) list of pairs separated by white space, or **%none**. One of the pairs may be written as **%defaultroute**, which means: find the interface *d* that the default route points to, and then act as if the value was ''**ipsec0=***d*''. **%defaultroute** is the default; **%none** must be used to denote no interfaces, or when using the NETKEY stack. If **%defaultroute** is used (implicitly or explicitly) information about the default route and its interface is noted for use by **ipsec_manual**(8) and **ipsec_auto**(8).)

listen
IP address to listen on (default depends on **interfaces=** setting). Currently only accepts one IP address.

nat_traversal

whether to accept/offer to support NAT (NAPT, also known as "IP Masqurade") workaround for IPsec. Acceptable values are: **yes** and **no** (the default). This parameter may eventually become per-connection.

disable_port_floating

whether to enable the newer NAT-T standards for port floating. Acceptable values are **no** (the default) and **yes** .

force_keepalive

whether to force sending NAT-T keep-alives to support NAT which are send to prevent the NAT router from closing its port when there is not enough traffic on the IPsec connection. Acceptable values are: **yes** and **no** (the default). This parameter may eventually become per-connection.

keep_alive

The delay (in seconds) for NAT-T keep-alive packets, if these are enabled using **force_keepalive** This parameter may eventually become per-connection.

virtual_private

contains the networks that are allowed as subnet= for the remote client. In other words, the address ranges that may live behind a NAT router through which a client connects. This value is usually set to all the RFC-1918 address space, excluding the space used in the local subnet behind the NAT (An IP address cannot live at two places at once). IPv4 address ranges are denoted as *%v4:a.b.c.d/mm* and IPv6 is denoted as *%v6:aaaa::bbbb:cccc:dddd:eeee/mm*. One can exclude subnets by using the **!**. For example, if the VPN server is giving access to 192.168.1.0/24, this option should be set to: *virtual_private=%v4:10.0.0.0/8,%v4:192.168.0.0/16,%v4:172.16.0.0/12,%v4:!192.168.1.0/24*. This parameter is only needed on the server side and not on the client side that resides behind the NAT router, as the client will just use its IP address for the inner IP setting. This parameter may eventually become per-connection.

oe

a boolean (yes/no) that determines if Opportunistic Encryption will be enabled. Opportunistic Encryption is the term to describe using IPsec tunnels without prearrangement. It uses IPSECKEY or TXT records to announce public RSA keys for certain IP's or identities.

For a complete description see /doc/draft-richardson-ipsec-opportunistic.txt, doc/opportunism-spec.txt and doc/opportunism.howto. See also the IETF BTNS working group and RFC4025.

The default is no.

This feature is only available with kernel drivers that support the caching of packets (%hold eroutes or equivalent) that allows us to respond to a packet from an unknown IP address. At present only the (klips)mast protocol stack supports this feature.

nhelpers

how many *pluto helpers* are started to help with cryptographic operations. Pluto will start *(n-1)* of them, where *n* is the number of CPU's you have (including hypherthreaded CPU's). A value of 0 forces pluto to do all operations in the main process. A value of -1 tells pluto to perform the above calculation. Any other value forces the number to that amount.

crlcheckinterval

interval, specified in seconds, after which pluto will verify loaded X.509 CRL's for expiration. If any of the CRL's is expired, or if they previously failed to get updated, a new attempt at updating the CRL is made. The first attempt to update a CRL is started at two times the crlcheckinterval. If set to **0**, which is also the default value if this option is not specified, CRL updating is disabled.

strictcrlpolicy

if not set, pluto is tolerant about missing or expired X.509 Certificate Revocation Lists (CRL's), and will allow peer certificates as long as they do not appear on an expired CRL. When this option is enabled, all connections with an expired or missing CRL will be denied. Active connections will be terminated at rekey time. This setup is more secure, but also dangerous. If the CRL is fetched through an IPsec tunnel with a CRL that expired, the entire VPN server will be dead in the water until a new CRL is manually transferred to the machine (if it allows non-IPsec connections). Acceptable values are **yes** or **no** (the default).

forwardcontrol

This option is obsolete and ignored. Please use **net.ipv4.ip_forward = 0** in /etc/sysctl.conf instead to control the ip forwarding behaviour.

rp_filter

This option is obsolete and ignored. Please use the **net.ipv4.conf/[iface]/rp_filter = 0** options in /etc/sysctl.conf instead. This option is badly documented; it must be 0 in many cases for ipsec to function.

syslog

the **syslog**(2) "facility" name and priority to use for startup/shutdown log messages, default **daemon.error**.

klipsdebug

how much KLIPS debugging output should be logged. An empty value, or the magic value **none**, means no debugging output (the default). The magic value **all** means full output. Otherwise only the specified types of output (a quoted list, names separated by white space) are enabled; for details on available debugging types, see **ipsec_klipsdebug**(8). This KLIPS option has no effect on NETKEY, Windows or BSD stacks.

plutodebug

how much Pluto debugging output should be logged. An empty value, or the magic value **none**, means no debugging output (the default). The magic value **all** means full output. Otherwise only the specified types of output (a quoted list, names without the **--debug-** prefix, separated by white space) are enabled; for details on available debugging types, see **ipsec_pluto**(8).

uniqueids

whether a particular participant ID should be kept unique, with any new (automatically keyed) connection using an ID from a different IP address deemed to replace all old ones using that ID. Acceptable values are **yes** (the default) and **no**. Participant IDs normally *are* unique, so a new (automatically-keyed) connection using the same ID is almost invariably intended to replace an old one.

plutorestartoncrash

prevent pluto from restarting after it crashed. This option should only be used when debugging a crasher. It will prevent overwriting a core file on a new start, or a cascade of core files. This option is also required if used with plutostderrlog= to avoid clearing the logs of the crasher. Values can be yes (the default) or no.

plutoopts

additional options to pass to pluto upon startup. See **ipsec_pluto**(8).

plutostderrlog

do not use syslog, but rather log to stderr, and direct stderr to the argument file.

pluto

whether to start Pluto or not; Values are **yes** (the default) or **no** (useful only in special circumstances).

plutowait

should Pluto wait for each negotiation attempt that is part of startup to finish before proceeding with the next? Values are **yes** or **no** (the default).

prepluto
: shell command to run before starting Pluto (e.g., to decrypt an encrypted copy of the *ipsec.secrets* file). It's run in a very simple way; complexities like I/O redirection are best hidden within a script. Any output is redirected for logging, so running interactive commands is difficult unless they use /dev/tty or equivalent for their interaction. Default is none.

postpluto
: shell command to run after starting Pluto (e.g., to remove a decrypted copy of the *ipsec.secrets* file). It's run in a very simple way; complexities like I/O redirection are best hidden within a script. Any output is redirected for logging, so running interactive commands is difficult unless they use /dev/tty or equivalent for their interaction. Default is none.

dumpdir
: in what directory should things started by *setup* (notably the Pluto daemon) be allowed to dump core? The empty value (the default) means they are not allowed to.

fragicmp
: whether a tunnel's need to fragment a packet should be reported back with an ICMP message, in an attempt to make the sender lower his PMTU estimate; acceptable values are **yes** (the default) and **no**. This KLIPS option has no effect on NETKEY, Windows or BSD stacks.

hidetos
: whether a tunnel packet's TOS field should be set to 0 rather than copied from the user packet inside; acceptable values are **yes** (the default) and **no**. This KLIPS option has no effect on NETKEY, Windows or BSD stacks.

overridemtu
: value that the MTU of the ipsec*n* **interface**(s) should be set to, overriding IPsec's (large) default. This parameter is needed only in special situations. This KLIPS option has no effect on NETKEY, Windows or BSD stacks.

Implicit Conns

The system automatically defines several conns to implement default policy groups. Each can be overridden by explicitly defining a new conn with the same name. If the new conn has **auto=ignore**, the definition is suppressed.

Here are the automatically supplied definitions.

```
conn clear
```

type=passthrough

authby=never

left=%defaultroute

right=%group

auto=route

conn clear-or-private
type=passthrough

left=%defaultroute

leftid=%myid

right=%opportunisticgroup

failureshunt=passthrough

keyingtries=3

ikelifetime=1h

salifetime=1h

rekey=no

auto=route
conn private-or-clear
type=tunnel

left=%defaultroute

leftid=%myid

right=%opportunisticgroup

failureshunt=passthrough

keyingtries=3

ikelifetime=1h

salifetime=1h

rekey=no

auto=route

conn private
type=tunnel

left=%defaultroute

leftid=%myid

right=%opportunisticgroup

failureshunt=drop

keyingtries=3

ikelifetime=1h

salifetime=1h

rekey=no

auto=route

conn block

type=reject

authby=never

left=%defaultroute

right=%group

auto=route

# default policy
conn packetdefault
type=tunnel

left=%defaultroute

leftid=%myid

left=0.0.0.0/0

right=%opportunistic

failureshunt=passthrough

keyingtries=3

ikelifetime=1h

salifetime=1h

rekey=no

auto=route

These conns are *not* affected by anything in **conn %default**. They will only work if **%defaultroute** works. The **leftid** will be the interfaces IP address; this requires that reverse DNS records be set up properly.

The implicit conns are defined after all others. It is appropriate and reasonable to use **also=private-or-clear** (for example) in any other opportunistic conn.

Policy Group Files

The optional files under /etc/ipsec.d/policy, including

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

may contain policy group configuration information to supplement *ipsec.conf*. Their contents are not security-sensitive.

These files are text files. Each consists of a list of CIDR blocks, one per line. White space followed by # followed by anything to the end of the line is a comment and is ignored, as are empty lines.

A connection in ipsec.conf which has **right=%group** or **right=%opportunisticgroup** is a policy group connection. When a policy group file of the same name is loaded, with

ipsec auto --rereadgroups

or at system start, the connection is instantiated such that each CIDR block serves as an instance's **right** value. The system treats the resulting instances as normal connections.

For example, given a suitable connection definition **private**, and the file /etc/ipsec.d/policy/private with an entry 192.0.2.3, the system creates a connection instance **private#192.0.2.3.** This connection inherits all details from **private**, except that its right client is 192.0.2.3.

Default Policy Groups

The standard Openswan install includes several policy groups which provide a way of classifying possible peers into IPsec security classes: **private** (talk encrypted only), **private-or-clear** (prefer encryption), **clear-or-private** (respond to requests for encryption), **clear** and **block**. Implicit policy groups apply to the local host only, and are implemented by the **IMPLICIT CONNECTIONS** described above.

CHOOSING A CONNECTION [THIS SECTION IS EXTREMELY OUT OF DATE

When choosing a connection to apply to an outbound packet caught with a **%trap,** the system prefers the one with the most specific eroute that includes the packet's source and destination IP addresses. Source subnets are examined before destination subnets. For initiating, only routed connections are considered. For responding, unrouted but added connections are considered.

When choosing a connection to use to respond to a negotiation which doesn't match an ordinary conn, an opportunistic connection may be instantiated. Eventually, its instance will be /32 -> /32, but for earlier stages of the negotiation, there will not be enough information about the client subnets to complete the instantiation.

Files

```
/etc/ipsec.conf
/etc/ipsec.d/policies/clear
/etc/ipsec.d/policies/clear-or-private
/etc/ipsec.d/policies/private-or-clear
/etc/ipsec.d/policies/private
/etc/ipsec.d/policies/block
```

See Also

ipsec(8), **ipsec_ttoaddr**(8), **ipsec_auto**(8), **ipsec_manual**(8), **ipsec_rsasigkey**(8)

History

Designed for the FreeS/WAN project <**http://www.freeswan.org**> by Henry Spencer.

Bugs

Before reporting new bugs, please ensure you are using the latest version of Openswan, and if not using KLIPS, please ensure you are using the latest kernel code for your IPsec stack.

When **type** or **failureshunt** is set to **drop** or **reject,** Openswan blocks outbound packets using eroutes, but assumes inbound blocking is handled by the firewall. Openswan offers firewall hooks via an "updown" script. However, the default **ipsec _updown** provides no help in controlling a modern firewall.

Including attributes of the keying channel (authentication methods, **ikelifetime**, etc.) as an attribute of a connection, rather than of a participant pair, is dubious and incurs limitations.

The use of **%any** with the *protoport*= option is ambiguous. Should the SA permits any port through or should the SA negotiate any single port through? The first is a basic conn with a wildcard. The second is a template. The second is the current behaviour, and it's wrong for quite a number of uses involving TCP. The keyword **%one** may be introduced in the future to separate these two cases.

ipsec_manual is not nearly as generous about the syntax of subnets, addresses, etc. as the usual Openswan user interfaces. Four-component dotted-decimal must be used for all addresses. It *is* smart enough to translate bit-count netmasks to dotted-decimal form.

It would be good to have a line-continuation syntax, especially for the very long lines involved in RSA signature keys.

First packet caching is only implemented for the KLIPS(NG) and MAST stacks. NETKEY returns POSIX-breaking responses, visiable as *connect: Resource temporarily unavailable* errors. This affects Opportunistic Encryption and DPD. Functionality on the BSD and Windows stacks is unknown.

Some state information is only available when using KLIPS, and will return errors on other IPsec stacks. These include *ipsec eroute*, *ipsec spi* and *ipsec look*.

Multiple L2TP clients behind the same NAT router, and multiple L2TP clients behind different NAT routers using the same Virtual IP is currently only working for the KLIPSNG stack.

The ability to specify different identities, **authby**, and public keys for different automatic-keyed connections between the same participants is misleading; this doesn't work dependably because the identity of the participants is not known early enough. This is especially awkward for the "Road Warrior" case, where the remote IP address is specified as 0.0.0.0, and that is considered to be the "participant" for such connections.

In principle it might be necessary to control MTU on an interface-by-interface basis, rather than with the single global override that **overridemtu** provides. This feature is planned for a future release.

A number of features which *could* be implemented in both manual and automatic keying actually are not yet implemented for manual keying. This is unlikely to be fixed any time soon.

If conns are to be added before DNS is available, **left=***FQDN*, **leftnextop=***FQDN*, and **leftrsasigkey=%dnsonload** will fail. **ipsec_pluto**(8) does not actually use the public key for our side of a conn but it isn't generally known at a add-time which side is ours (Road Warrior and Opportunistic conns are currently exceptions).

The **myid** option does not affect explicit **ipsec auto --add** or **ipsec auto --replace** commands for implicit conns.

Referenced By

ipsec_ipsec_pluto(8), **ipsec_showhostkey**(8), **strongswan**(8), **strongswan.conf**(5), **strongswan_ipsec.secrets**(5), **strongswan_openac**(8), **strongswan_pluto**(8)