

Advanced Features of Linux strongSwan the OpenSource VPN Solution



Andreas Steffen <andreas.steffen@hsr.ch>

Institute of Internet Technologies and Applications
Hochschule für Technik Rapperswil, Schweiz

The powerful advanced features of the Linux strongSwan VPN solution will be presented: IPsec policies based on wildcards, certificate hierarchies or group memberships defined by X.509 attribute certificates; certificate revocation based on the Online Certificate Status Protocol; Virtual IP address assignment; smartcard support; Dead Peer Detection. Also the new and user-friendly strongSwan User-Mode-Linux testing environment will be demonstrated.

1 IPsec-based Virtual Private Networks

Figure 1 shows a typical VPN scenario where two subnets 10.1.0.0/16 and 10.2.0.0/16 possessing private network addresses are connected with each other over the Internet by means of a *site-to-site* VPN tunnel. The tunnel is established between the VPN gateways 11.22.33.44 and 55.66.77.88 which automatically encrypt and authenticate each packet that is being exchanged between the two networks.

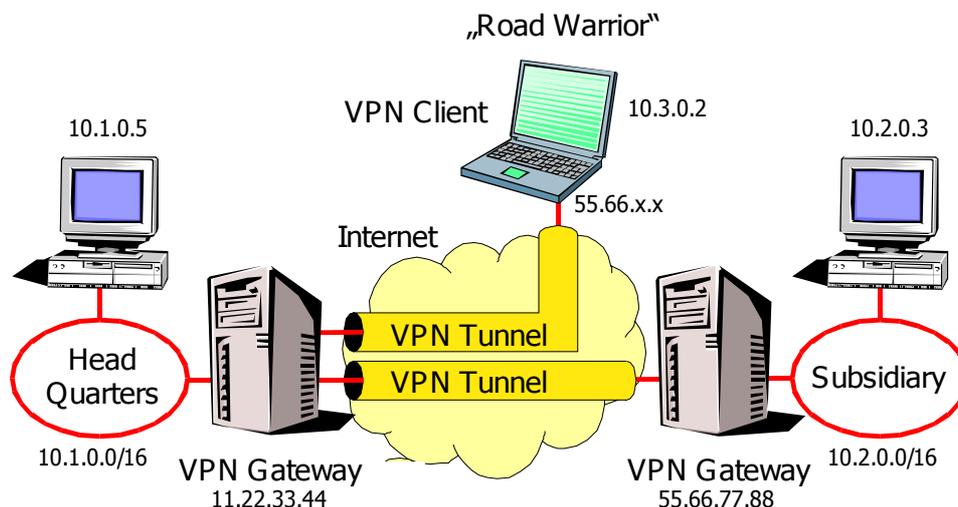


Figure 1: Typical VPN scenario

The second much more challenging scenario depicted in Figure 1 is *remote access*. So called „road warriors“, equipped with dynamical IP addresses assigned by their local Internet Service Providers (ISPs) are able to build up a VPN tunnel to the security gateway 11.22.33.44 from any point of the Internet using either fixed connections, public WLAN hot spots or mobile communication channels. Thus the remote access clients get full access to all resources in the 10.1.0.0/16 network as if they were located right at Head Quarters.

We will come back to the special properties of the road warrior case later in this paper and continue by first giving a short overview on the IPsec standard which basically consists of two parts:

- The actual encryption and authentication of tunneled IP packets takes place in the **kernel** using the *Encapsulating Security Payload* (ESP) standard defined by RFC 2406. ESP is IP **protocol 50** and doesn't have ports.
- The initial negotiation and the ensuing periodic re-keying of IPsec tunnels is done by a **userland daemon** running the *Internet Key Exchange* (IKE) protocol defined by RFCs 2407, 2408, and 2409. IKE is transported over UDP datagrams and must use the well-known source and destination **port 500**.

1.1 The IPsec Kernel Part - ESP

The ESP encapsulation of IP packets is shown in figure 2. Referring to the site-to-site VPN scenario depicted in figure 1, a TCP packet exchanged between a host 10.1.0.5 in the 10.1.0.0/16 network and a host 10.2.0.7 in the 10.2.0.0/16 network will carry these two addresses as source and destination in the original IP header, whereas the outer IP header of the IPsec packet will contain the addresses 11.22.33.44 and 55.66.77.88 of the VPN gateways that do the actual tunneling. ESP encrypts the whole original IP packet including the internal header and secures the encrypted content against unauthorized modification by computing and appending a cryptographic checksum.

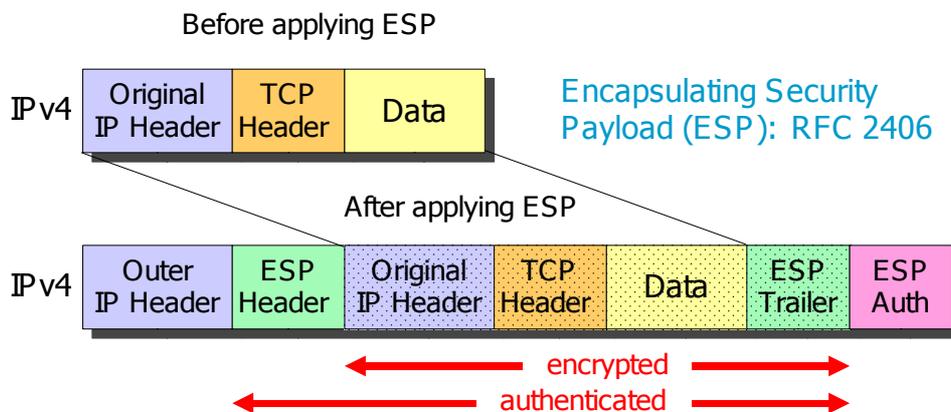


Figure 2: IPsec Encapsulating Security Payload (ESP)

Depending on the Linux kernel version, strongSwan employs different mechanisms to implement the IPsec kernel part:

- Under a Linux 2.4 kernel, **KLIPS** from the FreeS/WAN project is used to implement the IPsec kernel functionality. KLIPS, distributed as part of strongSwan, can either be compiled statically into the kernel or loaded dynamically as a kernel module *ipsec.o*. For encryption, authentication and compression, built-in functions from the FreeS/WAN project and/or crypto modules provided by Juanjo Ciarlante are used.
- Under a Linux 2.6 kernel, the native **KAME** stack ported from the BSD project to Linux is used. Encryption, authentication and compression tasks use the standard modules offered by the kernel's crypto API.

1.2 The IPsec Userland Part - IKE

strongSwan's userland daemon **pluto** is responsible for setting up, re-keying and deleting IPsec tunnels using the standardized Internet Key Exchange (IKE) protocol. An IKE negotiation is divided into *Phase 1* where the VPN peers do mutual authentication, followed by one or several *Phase 2* exchanges where the encryption, authentication and compression parameters for the actual tunneling of IP packets between predefined subnets are set up.

strongSwan implements *IKE Main Mode* for Phase 1 and *IKE Quick Mode* for Phase 2. The potentially vulnerable *IKE Aggressive Mode* Phase 1 variant is not supported by strongSwan out of security considerations. IKE Main Mode peer authentication can either be based on Pre-Shared Keys (PSK) or on RSA signatures as shown in figure 3.

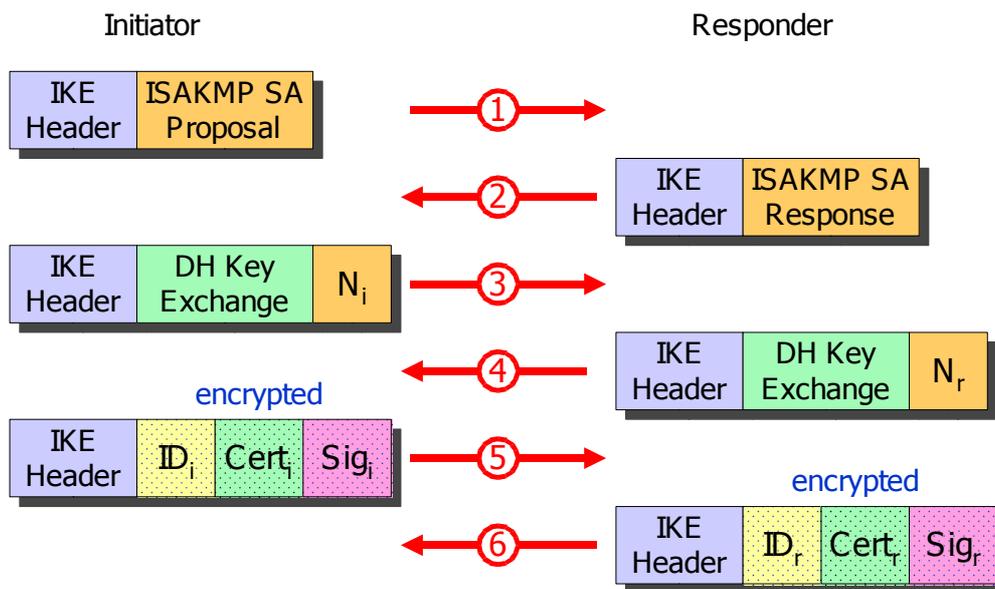


Figure 3: Internet Key Exchange Protocol (IKE)

IKE Main Mode consists of six messages exchanged between the VPN peers:

1. The initiator proposes a series of supported encryption and authentication transforms for securing the IKE negotiation.
2. The responder selects a set of transforms common to both parties.
3. The initiator sends a public Diffie-Hellman factor and a nonce.
4. The responder in turn also sends a public Diffie-Hellman factor and a nonce which is a random number. Using the Diffie-Hellman key exchange algorithm both end points can now compute a common shared secret that is used to encrypt all ensuing IKE messages.
5. The initiator sends its identity and a signature computed by encrypting a hash formed over all exchanged IKE messages with its RSA private key. Although an option in IKE, strongSwan always includes a trusted X.509 certificate that can be used by the peer to verify the signature.
6. Now it is the responder's turn to identify and authenticate itself.

Each IKE Quick Mode will then add another three messages.

2 From FreeS/WAN to strongSwan

The **FreeS/WAN** project (www.freeswan.org) was founded in 1999 by John Gilmore with the ultimate goal of automatically encrypting a significant part of the Internet traffic using *Opportunistic Encryption* (OE) based on IPsec and IKE. The grand idea behind OE was to do host authentication using raw RSA public keys fetched via the ubiquitous Domain Name System (DNS).

Because most existing VPN implementations did not and still do not support the use of raw RSA keys, the author decided to contribute a X.509 patch to the FreeS/WAN project in order to make it possible for Linux hosts to set up IPsec tunnels with any other VPN product using standardized X.509 certificates. The first X.509 patch was released in 2000 and all further versions delivered over the next four years were developed by the author as professor for security and communications at the Zürcher Hochschule Winterthur (ZHAW) with major contributions from a whole group of diploma students.

In 2002, due to the increasing demand for the X.509 patch, Ken Bantoft bundled it with several other FreeS/WAN add-ons like Mathieu Lafon's NAT traversal patch and Juanjo Ciarlante's alternative crypto algorithms and started his **Super FreeS/WAN** distribution that quickly became extremely popular.

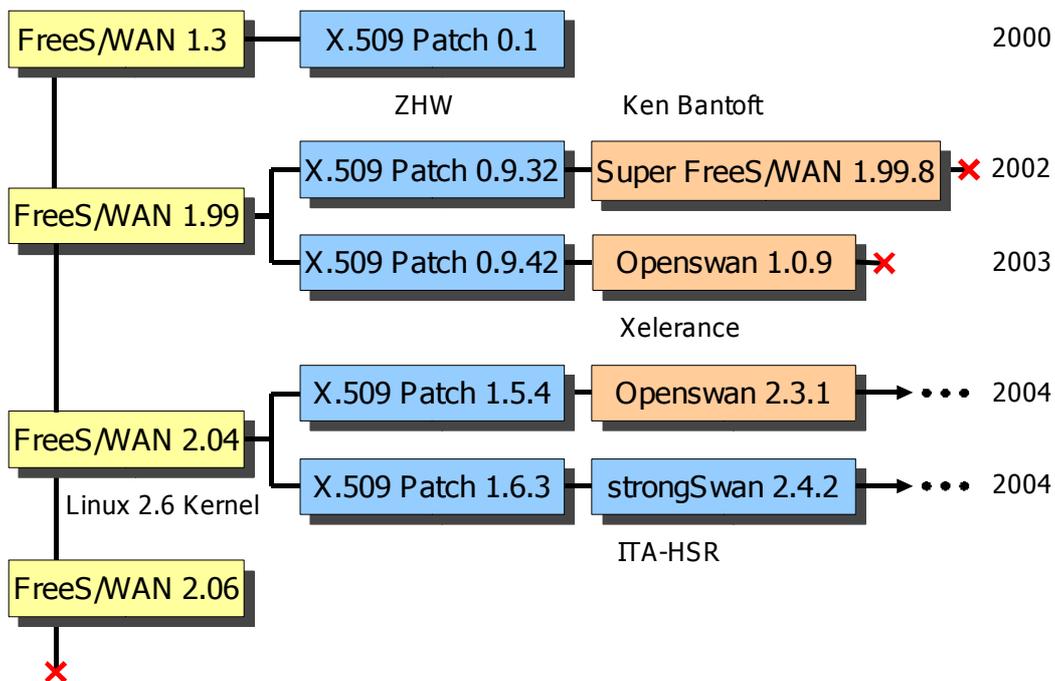


Figure 4: The FreeS/WAN Genealogy

Towards the end of 2003 when it became evident that the FreeS/WAN project was going to be discontinued in spring 2004 with the final 2.06 release, Ken Bantoft, FreeS/WAN project leader Michael Richardson and Paul Wouters founded Xelerance Corporation with the goal of carrying on the IPsec development within their **Openswan** project (www.openswan.org).

Whereas Openswan has been moving closer to the mainstream VPN path by adding IKE Aggressive Mode and Cisco's legacy XAUTH authentication, the author decided to fork a **strongSwan** distribution (www.strongswan.org) of his own in order to be able to quickly integrate and deploy new certificate-based features originating from the Zürcher Hochschule Winterthur.

In March 2005 the author accepted an offer to join the Hochschule für Technik Rapperswil (HSR, www.hsr.ch) where as a professor for security and communications he is now heading the Institute for Internet Technologies and Applications (ITA). Being an important part of the ITA strategy, the maintenance and continuing evolution of the strongSwan distribution will be guaranteed in the years to come.

3 The „Road Warrior“ Remote Access Case

One of strongSwan's powerful features inherited from FreeS/WAN is the support of road warrior connections as shown in figure 5.

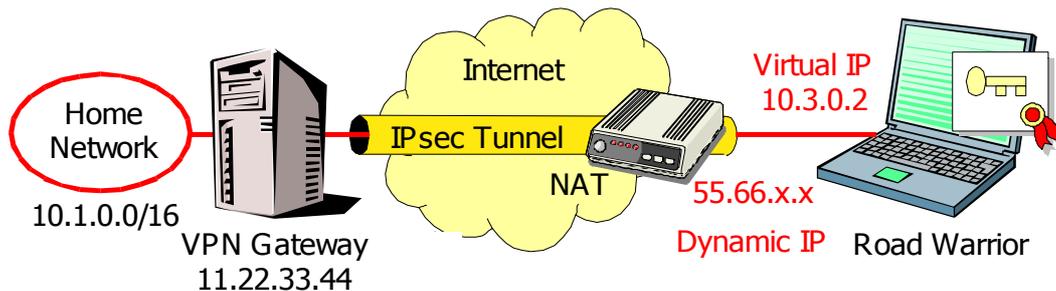


Figure 5: Specific properties of the road warrior scenario

The following three properties make the remote access case special:

- The IP address of a road warrior is nearly always *dynamic*, i.e. it is usually assigned by a local ISP, so that a VPN gateway cannot get any information on the identity of the remote-access client by looking at the source address.
- Since IKE Main Mode with Pre-Shared Keys (PSK) does not work with dynamic addresses and the workaround based on IKE Aggressive Mode is insecure, *X.509 certificates* should be used to establish the identity of a road warrior.
- In order to ensure that IP packets originating from the home network find the way back through the IPsec tunnel and are not mistakenly routed directly into the Internet via the default gateway, the inner IP address used in the tunnel should be set to a *virtual address* taken from a special remote-access pool instead of being equal to the road warrior's dynamic outer address.

The template in figure 6 defines a road warrior connection on a VPN gateway:

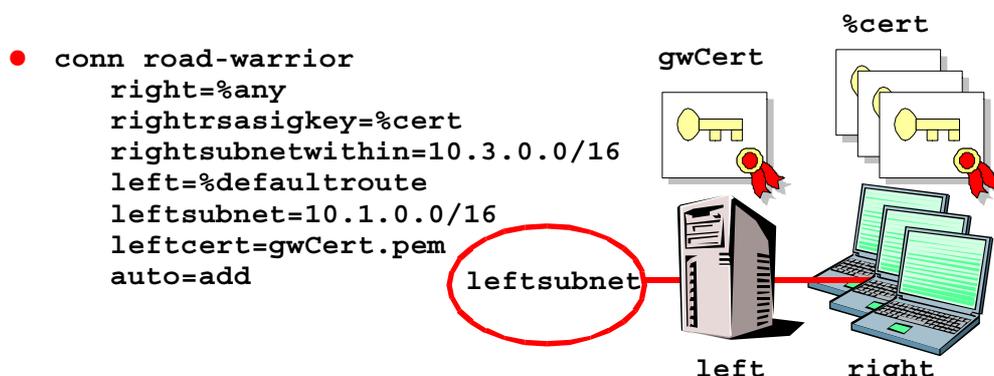


Figure 6: Road warrior connection definition on strongSwan gateway

- *right=%any* takes into account that the IP address of the incoming road warrior is dynamic and therefore a priori unknown.
- *rightrsasigkey=%cert* signifies that the peer's RSA public key will be made available embedded in a X.509 certificate. This means that any road warrior presenting a certificate issued by a Certification Authority (CA) the VPN gateway puts trust in, will be allowed to set up an IPsec connection. Thus an unlimited number of road warrior instances can be derived from this single template.
- *rightsubnetwithin=10.3.0.0/16* defines an address range within which all virtual peer addresses must lie.
- *left=%defaultroute* assigns the IP address of the default network interface to the VPN gateway.
- *leftsubnet=10.1.0.0/16* defines the internal network hidden behind the VPN gateway.
- *leftcert=gwCert.pem* designates the path to the VPN gateway's own certificate.
- *auto=add* means the connection definition is loaded into memory when strongSwan is started up. The keying daemon pluto then waits passively for incoming road warrior connections.

In all our examples we will use the convention that **left** will designate the **local** side and **right** the **remote** side of a VPN tunnel although strongSwan would also allow to define the directions the other way round.

3.1 Virtual IP Address Assignment

On a strongSwan road warrior a virtual IP address can be assigned statically using the **leftsourceip** statement:

```
conn home
    right=11.22.33.44           # IP of VPN gateway
    rightid=@gateway.kool.net  # ID of VPN gateway
    rightsubnet=10.1.0.0/24    # subnet behind gateway
    left=%defaultroute         # dynamic external IP
    leftsourceip=10.3.0.2      # static virtual IP
    leftcert=bodoCert.pem      # Bodo's certificate
    leftid=bodo@kool.net       # Bodo's ID
    auto=start                  # start tunnel automatically
```

In large VPNs with many users it would be preferable if the virtual addresses could be retrieved from a centralized store and be pushed down to the road warriors via the VPN gateway. This can be realized by means of the IKE *Mode Config* protocol. Thus with dynamic assignment of the virtual IP address the configuration on the road warrior changes to

```
conn home
    . . . .
    leftsourceip=%modeconfig  # virtual IP assigned dynamically
    auto=start
```

On the VPN gateway which will act as a *Mode Config server*, the virtual IP addresses must currently be defined for each road warrior in *ipsec.conf* using the **rightsourceip** parameter. Thus the definition from figure 6 changes to

```
conn %default
    right=%any
    left=%defaultroute
    leftsubnet=10.1.0.0/16
    leftcert=gwCert.pem
    leftid=@gateway.kool.net
    auto=add

conn antje
    rightid=antje@kool.net
    rightsourceip=10.3.0.1

conn bodo
    rightid=bodo@kool.net
    rightsourceip=10.3.0.2

conn ...
```

Future versions of strongSwan will allow for the virtual IP addresses to reside on an LDAP server which will further facilitate the management of large VPNs.

3.2 Dead Peer Detection

Dead Peer Detection (DPD, RFC 3706) is another useful feature implemented by strongSwan. DPD avoids dangling IPsec security associations with peers that suddenly vanish without properly terminating their tunnels via IKE Delete SA notifications. Figure 7 shows the timing diagram of the DPD protocol.

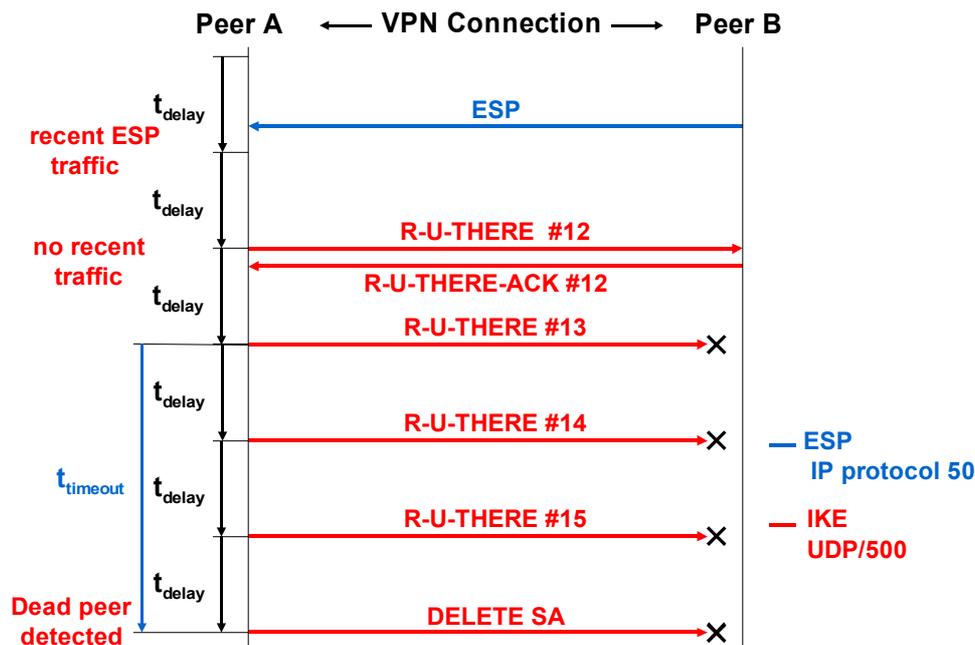


Figure 7: Dead Peer Detection (DPD)

When DPD is activated as in the following connection definition:

```
conn road-warrior
    right=%any
    ...
    dpddelay=1m           # check connection every minute
    dpdtimeout=3m        # timeout after 5 minutes
    dpdaction=clear      # clear connection after timeout
    auto=add
```

then strongSwan checks every **dpddelay** interval if any ESP traffic has been received from the peer. If this has not been the case then a R-U-THERE IKE notification message is sent to the peer who replies with a R-U-THERE-ACK keep-alive message. If no acknowledgement is received over a **dpdtimeout** interval then all IPsec tunnels with the dead peer are automatically cleared.

3.3 Smartcard Support

Storing the RSA private key on a smartcard or USB crypto token as shown in figure 9 minimizes the risk in the case of theft or loss of a portable laptop computer.



Figure 8: Support of smartcards and USB crypto tokens

strongSwan offers a standardized PKCS #11 crypto token interface that can be used either with the OpenSC smartcard library (www.opensc.org) or any other third party PKCS #11 module.

A certificate stored on a smartcard e.g. under the object ID 52 can be referenced with the command

```
conn home
    ...
    leftcert=%smartcard:52
    leftid=bodo@kool.net
    auto=add
```

Since the private RSA key stored on the crypto token is protected by a PIN, the statement

```
: PIN %smartcard:52 %prompt
```

in `/etc/ipsec.secrets` will cause pluto to prompt for the PIN code when the connection `home` is started on the laptop computer.

In the upcoming 2.4.2 release, strongSwan is going to support card readers equipped with a PIN pad as depicted in figure 10. Using the key pad and the display of the secure smartcard reader as IO devices it will be possible to start and stop VPN connections without the need for an additional keyboard.



Figure 9: PIN-pad-controlled strongSwan security gateway

4 Certificate Revocation Mechanisms

If user authentication in a VPN is based on X.509 certificates than it becomes of utmost importance that certificates can be quickly revoked e.g when the corresponding RSA private key gets compromised or if the user loses the right to access the VPN. In this chapter we are going to present two supported revocation methods , namely the dynamic download of Certificate Revocation Lists (CRLs) via HTTP or LDAP Uniform Resource Identifiers (URIs) and the Online Certificate Status Protocol (OCSP).

4.1 Certificate Revocation Lists

The most elegant way to define one or several CRL Distribution Points (CDPs) in the form of HTTP or LDAP URIs, is to put them as X.509v3 certificate extensions right into the host or user certificates. This can be achieved during certificate generation with the following OpenSSL configuration:

```
crlDistributionPoints = # HTTP URI
    URI:http://crl.kool.net/cert.crl

crlDistributionPoints = # LDAP URI
    URI:ldap://ldap.kool.net/o=Kool AG,c=CH
    ?certificateRevocationList?base
    ?(objectClass=certificationAuthority)
```

Whenever a certificate is received from the peer via the IKE protocol, then strongSwan will automatically start a thread to download the CRL. From then on a watchdog thread will periodically check if an update is available and will fetch it shortly before the old CRL expires.

If no built-in CDP extensions are present in the certificates or if additional CDPs become available, the URIs can alternatively be defined for each certification authority in a special **ca** section in *ipsec.conf*.

```

ca kool
  cacert=koolCA.pem
  crluri=http://crl.kool.net/cert.crl
  crluri2=http://crl2.kool.net/cert.crl
  ocsपुरi=http://ocsp.kool.net:8880
  auto=add

```

The ca section above defines two CRL URIs plus an URI pointing to an OCSP server, the functionality of which we are going to discuss next.

4.2 Online Certificate Status Protocol

An OCSP server can be queried in near-real time about the current status of a given certificate using the Online Certificate Status Protocol (OCSP, RFC 2560). OCSP employs a HTTP-based request/response scheme as shown in figure 10.

An OCSP request must contain the issuer and the serial number of the certificate in question and can be optionally signed by the requestor. The OCSP response takes on one of the values: *good*, *revoked*, or *unknown*, and is signed by the OCSP server.

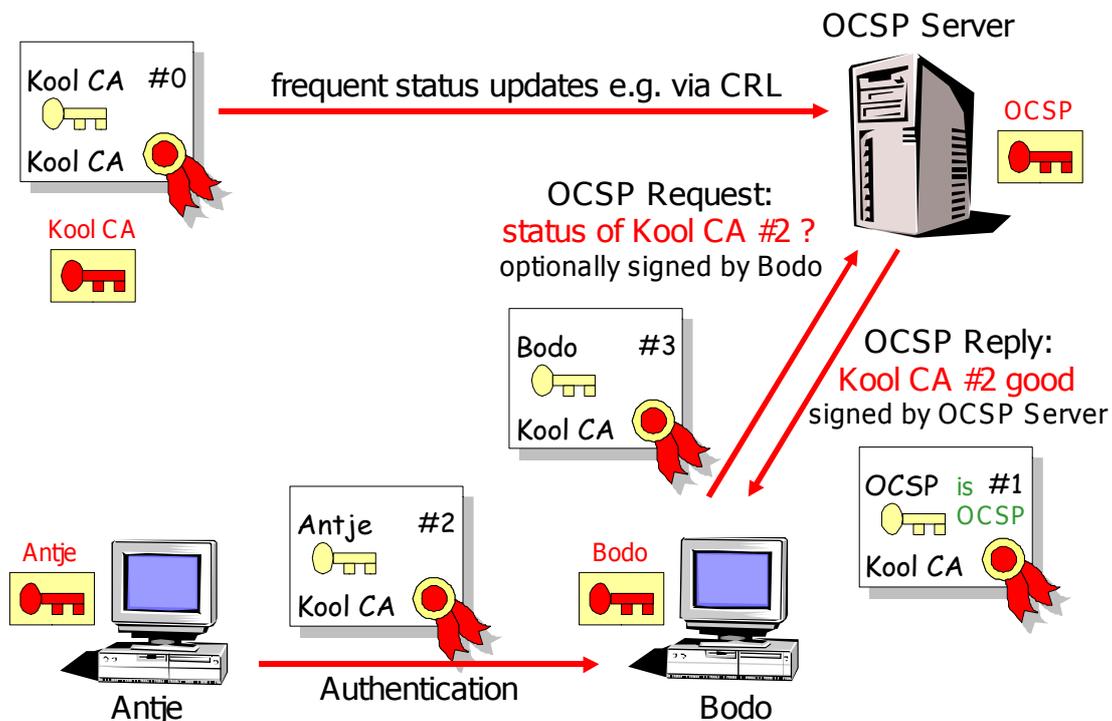


Figure 10: Online Certificate Status Protocol (OCSP)

Trust into the OCSP signer is established either by manually importing the corresponding OCSP signing certificate into the `/etc/ipsec.d/ocspcerts/` directory or by looking for the `extendedKeyUsage=OCSPSigner` flag in the OCSP certificate issued by the CA responsible for the revocation information. In the latter case the OCSP signing certificate can be distributed by the server itself by including it in the OCSP response.

5 Advanced IPsec Policies

strongSwan's most powerful feature is the support of sophisticated IPsec policies based on either wildcard parameters, certificate hierarchies or attribute certificates. In this section we will present all three principles.

5.1 Based on Wildcard Parameters

With the road warrior connection definition of figure 6 any peer possessing a trusted certificate can access the subnet protected by the security gateway. If we want to restrict the access to specific user groups then we must define a corresponding *IPsec policy*. This can be done by defining pattern matching rules operating on the identity of the users. A solution recommended by RFC 3586 *IPsec Policy Information Model* is to apply wildcards to the subject distinguished name of the user certificates. In the example of figure 11 the research network 10.1.1.0/24 can be accessed by anyone (CN=*) belonging to the R&D department (OU=R&D), whereas the sales network 10.1.2.0/24 is open to the sales staff (OU=Sales), only .

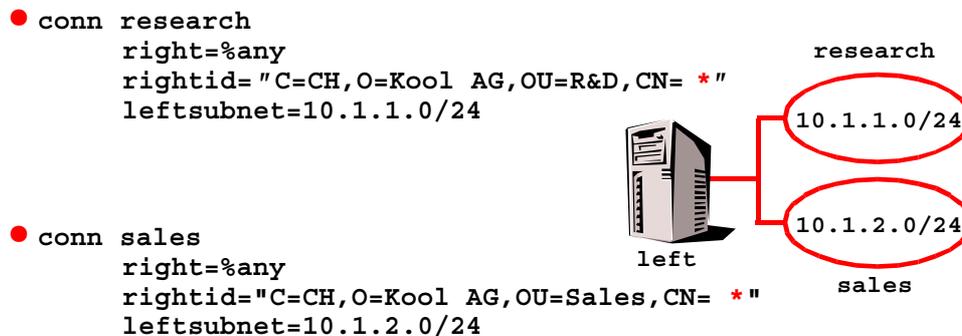


Figure 11: IPsec policy based on wildcards

5.2 Based on Certification Authorities

Another approach of dividing the peers into specific user groups is the creation of intermediate certification authorities as shown in figure 12.

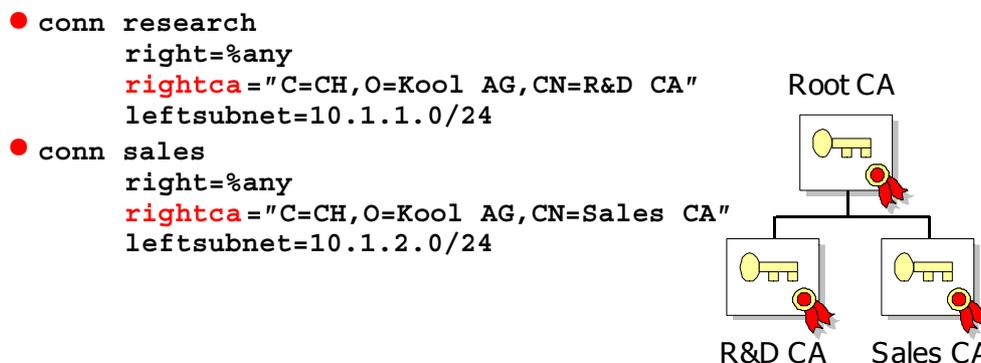


Figure 12: IPsec policy based on certification authorities

In this scenario both the research and sales departments issue user certificates of their own. The intermediate certification authorities *R&D CA* and *Sales CA*, respectively, are in turn certified by a common *Root CA*. Access to the departmental networks are now restricted to the matching CA by using the *rightca* parameter.

5.3 Based on X.509 Attribute Certificates

The most flexible policy approach is based on group memberships certified by X.509 *Attribute Certificates* that are issued by an *Authorization Authority*. Figure 13 shows the relationship between user and attribute certificates as well as between certification and authorization authorities.

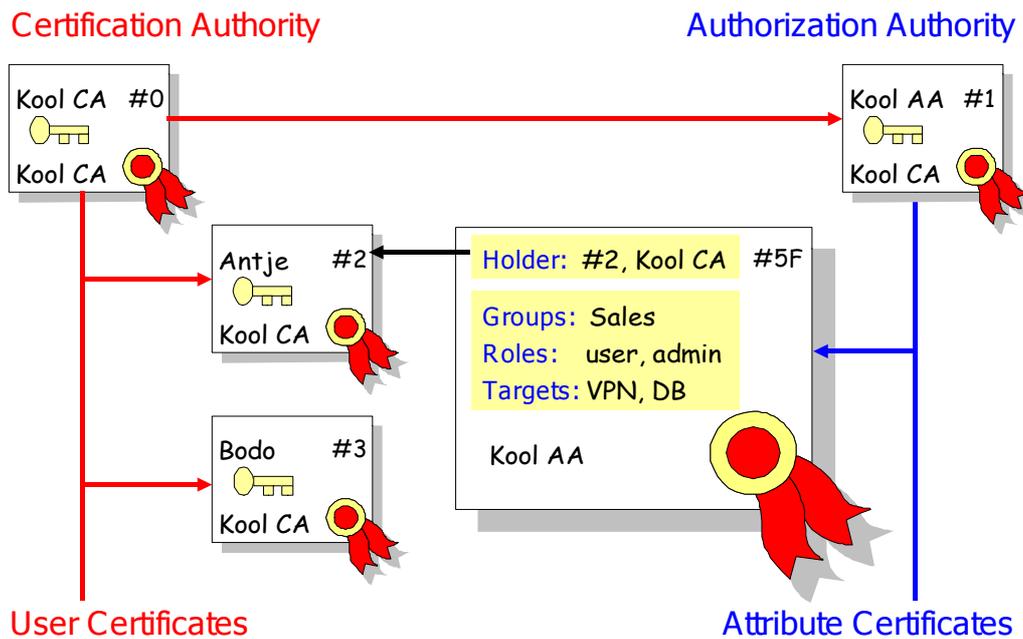


Figure 13: X.509 attribute certificates

A certification authority issues long-lived user certificates that ideally contain only information fields that will rarely change. All short-lived personal attributes that may include group memberships, roles, target systems, time profiles, billing information, etc. are put into special attribute certificates that are issued by a trusted authorization authority. Each attribute certificate is linked to its holder by including the issuer and serial number of the user certificate in the attribute certificate. Attribute certificates can be issued e.g. on a daily basis, so that the user access profiles will always be up-to-date and because of the short validity interval there will be no need for a revocation mechanism.

strongSwan supports the implementation of IPsec policies based on group memberships as shown in the example of figure 14. The **rightgroups** parameter is used to enumerate the groups that are allowed to access the network resources. Thus in our example, members of the group Research can connect to the research network, only, whereas the members of either the Accounting or Sales group are given exclusive access to the Sales network.

- `conn research`
`right=%any`
`rightgroups=Research`
`leftsubnet=10.1.1.0/24`
- `conn sales`
`right=%any`
`rightgroups="Accounting, Sales"`
`leftsubnet=10.1.2.0/24`

Figure 14: IPsec policy based on group memberships

The **openac** program is a part of the strongSwan distribution and can be used to generate X.509 attribute certificates linked to given users. Currently only group attributes are supported and the generated certificates must be copied into the `/etc/ipsec.d/aacerts/` directory from where they are loaded by pluto. In future strongSwan releases it will become possible to automatically fetch the attribute certificates from an LDAP server.

6 User-Mode-Linux Testing Environment

The UML testing environment for strongSwan was created by Eric Marchionni and Patrik Rayo (both recent graduates from the Zürcher Hochschule Winterthur, Switzerland). Details on the implementation can be found in their diploma thesis (http://home.zhwin.ch/~sna/DA/Sna3_2004.pdf). Although a UML test suite originally written by Michael Richardson already existed for the FreeS/WAN 2.04 distribution, the students decided to start from scratch in order to make the environment more user-friendly.

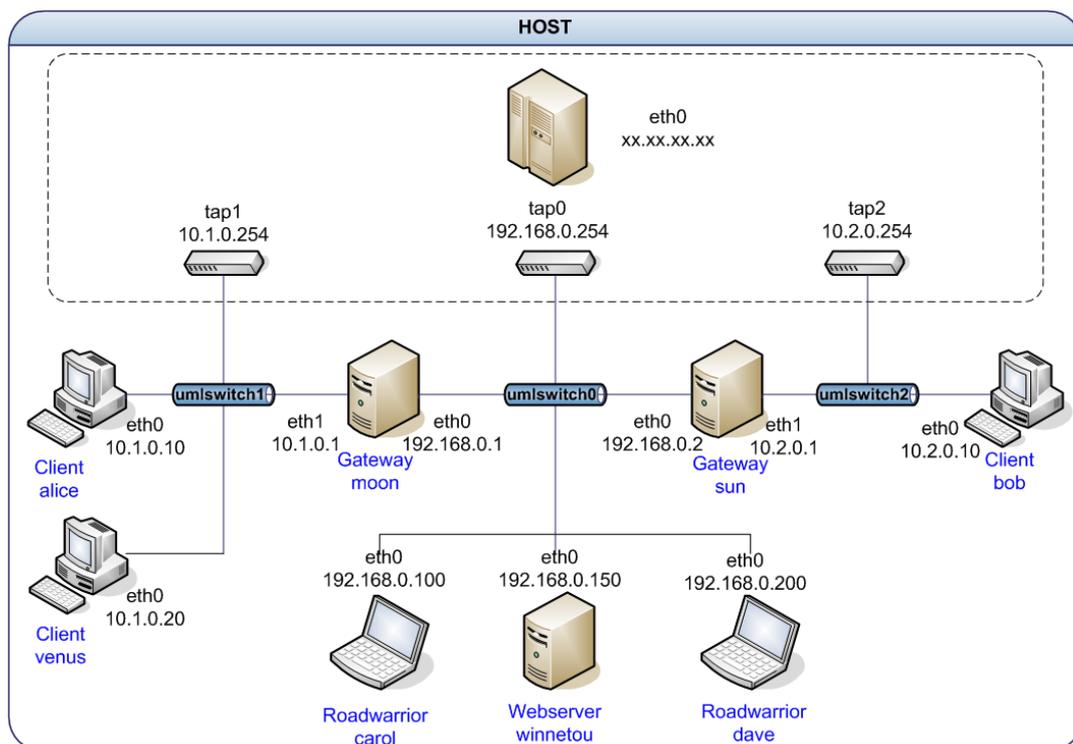


Figure 15: strongSwan UML network topology

Figure 15 shows the default UML network topology created for the strongSwan testing environment. It consists of a maximum of eight virtual hosts: the gateways *moon* and *sun* guarding the subnets 10.1.0.0/16 and 10.2.0.0/16, respectively; the clients *alice*, *carol*, and *bob* populating these subnets; the road warriors *carol* and *dave*; and finally *winnetou* used as a HTTP server. Three tun/tap devices connect the host system with each of the UML instances via the UML switches sitting at the center of the corresponding sub-networks.

If not all of the eight hosts are needed for a given simulation scenario then the desired instances can be started by enumerating them on the command line:

```
start-testing alice moon carol winnetou
```

6.1 Interactive Mode

The most flexible way to use the strongSwan UML environment is the interactive mode shown in figure 16. On a graphical desktop either a KDE *konsole* or an *xterm* is opened for each started instance. It is also possible to open a terminal console on the host system via remote access and switch between the various UML instances using the *screen* command.

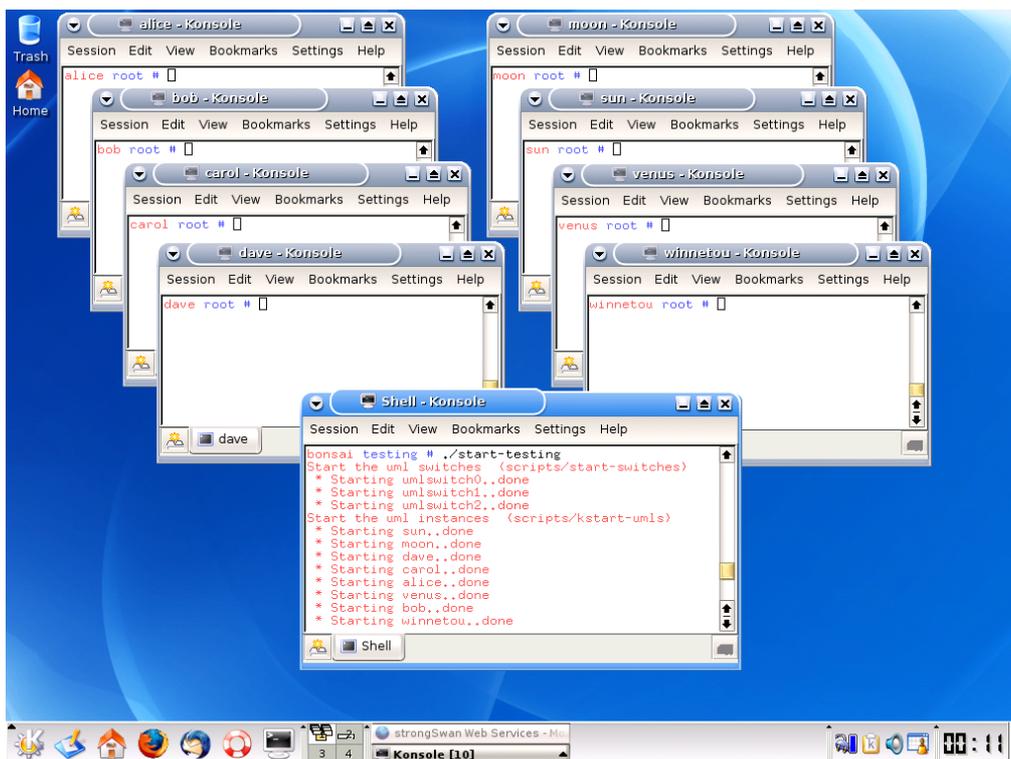


Figure 16: strongSwan UML network in interactive mode

The interactive mode is ideally suited for debugging new strongSwan releases because all communication signals exchanged *between* the hosts as well as all log files and debug information *on* the hosts are fully available in a controlled environment. By including *gcc*, *gdb* and *tcpdump* in the UML root file system, code can be modified, recompiled and tested on the fly right on the UML instances. In November 2004, using the interactive mode it was possible to reproduce a reported IKE re-keying problem occurring in the presence of NAT after two hours of UML simulation and to release a bug fix on the same day!

6.2 Automated Software Regression Test Mode

Software regression tests are run prior to each new strongSwan release in order to verify the compliance with the specifications and also to detect bugs in an early stadium. Since the UML interactive mode is too error prone and too tedious because of the manual configuration steps involved, the diploma students Eric Marchionni and Patrik Rayo also created an automated testing framework for strongSwan.

A regression test suite consists of a large number of scenarios that are executed automatically and the test results are analyzed without manual intervention. The strongSwan testing framework creates a subdirectory for each scenario as the example in figure 17 demonstrates.

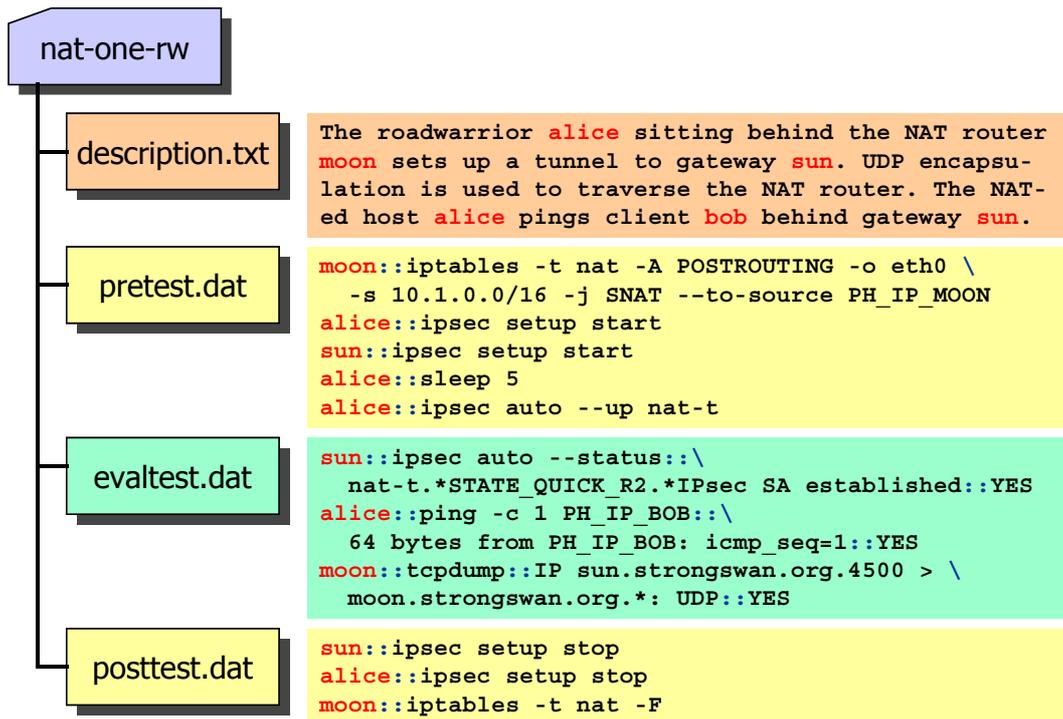


Figure 17: Scripts for automated software regression testing

The file *description.txt* gives a concise summary of the scenario. The next file *pretest.dat* contains a list of commands that are executed sequentially on the various UML instances used by the given scenario. In our example a NAT rule is inserted on the router *moon*. Next the ipsec daemon is started on the VPN end points *alice* and *sun*. A sleep command of 5 seconds makes sure that both daemons will be up before the last command is executed which builds up the NAT-ed IPsec connection using the Internet Key Exchange protocol (IKE).

In a second phase the commands of the file *evaltest.dat* are executed which by applying pattern matching rules evaluate if the desired test results have been achieved. In our example it is first checked if the IKE negotiation has been successful both on *sun* and *alice*. Next a ping from *alice* to *bob* executed on *alice* checks the connectivity through the NAT-ed IPsec tunnel. The last check on router *moon* verifies if the standardized UDP port 4500 has been used for the NAT traversal.

In the third and last phase the commands in *posttest.dat* reset all UML instances involved in the scenario to the idle state at the outset of the test. This is achieved by stopping *ipsec* on the VPN peers and by flushing the router's NAT rule.

The three phases *pretest*, *evaltest* and *posttest* are controlled by a script running on the host system. The commands are executed on the various UML hosts using *ssh* (secure shell), e.g.

```
ssh root@alice ipsec setup start
```

At the end of each test a selection of log and status files is copied from the UML instances back to the host system using *ssh* and *scp* (secure copy).

6.3 Display of Test Results

For each test scenario a HTML page is automatically created and copied together with the most relevant configuration, log and status data to the UML web server *winnetou* that has by default the IP address 192.168.0.150.

The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://192.168.0.150/testresults/20050428-0046/nat-one-rw/`. The page content includes:

- Test nat-one-rw**
- Description**

The roadwarrior **alice** sitting behind the NAT router **moon** sets up a tunnel to gateway **sun**. UDP encapsulation is used to traverse the NAT router. In order to test the tunnel, the NAT-ed host **alice** pings the client **bob** behind the gateway **sun**.
- Files:**
 - [console.log](#)
- Network Diagram:**

The diagram shows a network topology with the following components and connections:

 - Client alice** (IP: 10.1.0.10) connected to **umlswitch1** (eth0).
 - Gateway moon** (IP: 10.1.0.1) connected to **umlswitch1** (eth1).
 - Gateway moon** (eth0, IP: 192.168.0.1) connected to **umlswitch0** (eth0).
 - Gateway sun** (IP: 192.168.0.2) connected to **umlswitch0** (eth0).
 - Gateway sun** (eth1, IP: 10.2.0.1) connected to **umlswitch2** (eth0).
 - Client bob** (IP: 10.2.0.10) connected to **umlswitch2** (eth0).
 - Webserver winnetou** (IP: 192.168.0.150) connected to **umlswitch0** (eth0).
- alice**
 - [ipsec.conf](#)
 - [ipsec.secrets](#)
 - [ipsec auto --statusall](#)
 - [ipsec auto --listall](#)
 - [auth.log](#)

Figure 18: Test results for NAT-T scenario published on winnetou

As the sample screen shot in figure 18 shows, all relevant information pertinent to a given test scenario can be conveniently accessed and examined using a standard web browser, without the need to configure a web server on the host system itself. Just do not forget to include *winnetou* in the list of started UML instances!

The next web page depicted in figure 19 is located one hierarchy level higher and aggregates the results from the individual tests. Currently 35 tests covering various strongSwan features have been defined. Depending on the hardware of the underlying host system a full automated test run takes between 30-60 minutes. With one glance it can then be verified if a software release has passed all regression tests.

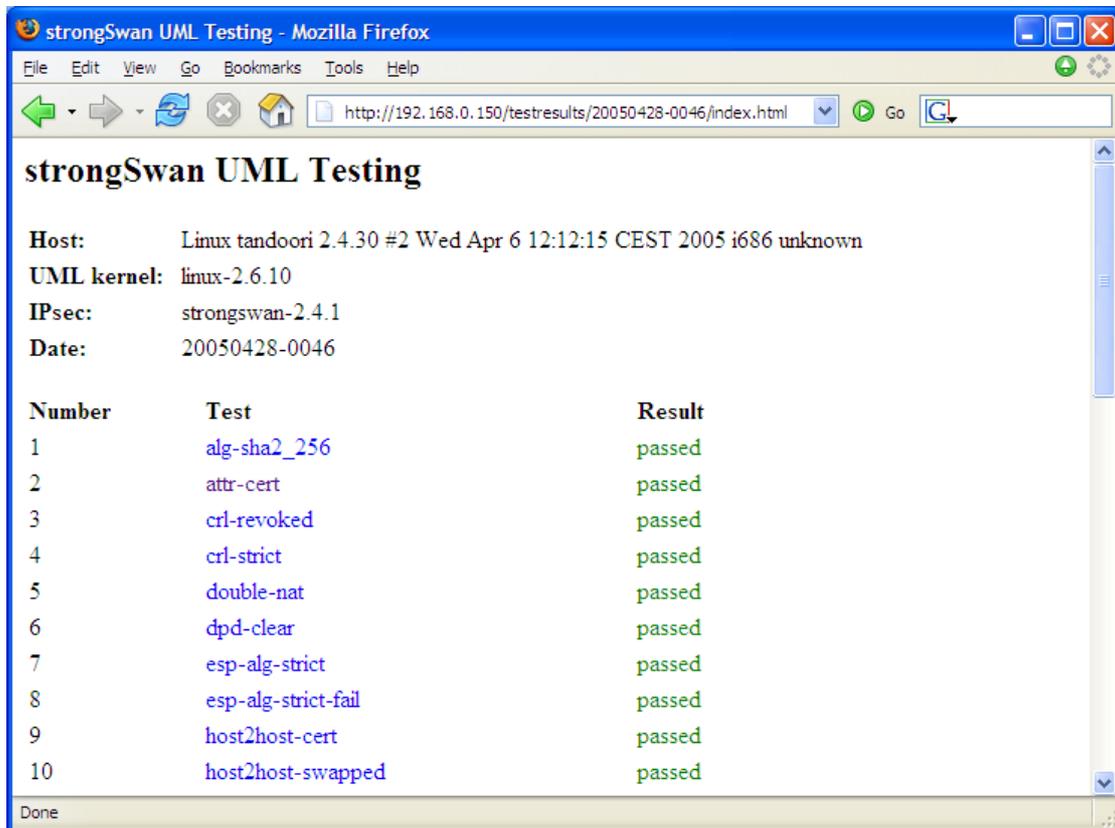


Figure 19: Overview on completed regression tests

The latest strongSwan scenarios are available from www.strongswan.org/uml/.

7 Conclusions

In a short tour we have presented to you the advanced features of the Linux strongSwan IPsec solution. In our opinion, the strongSwan distribution unfolds its full strength in a certificate-based public key infrastructure environment, preferably augmented by a smartcard-based private key management.

A unique feature offered by strongSwan is the possibility to define sophisticated IPsec policies based on group attributes that can be securely deployed by putting them into in short-lived X.509 attribute certificates. In future strongSwan releases we want to extend this capability.

Since its first release in January 2005, the on-line collection of strongSwan test scenarios has been visited by many people looking for solutions to their VPN setup problems. The interactive UML simulation mode is also an excellent tool for exploring various VPN scenarios.