# Virtual Private Networks using *BSD
## - a case study -

**Index**

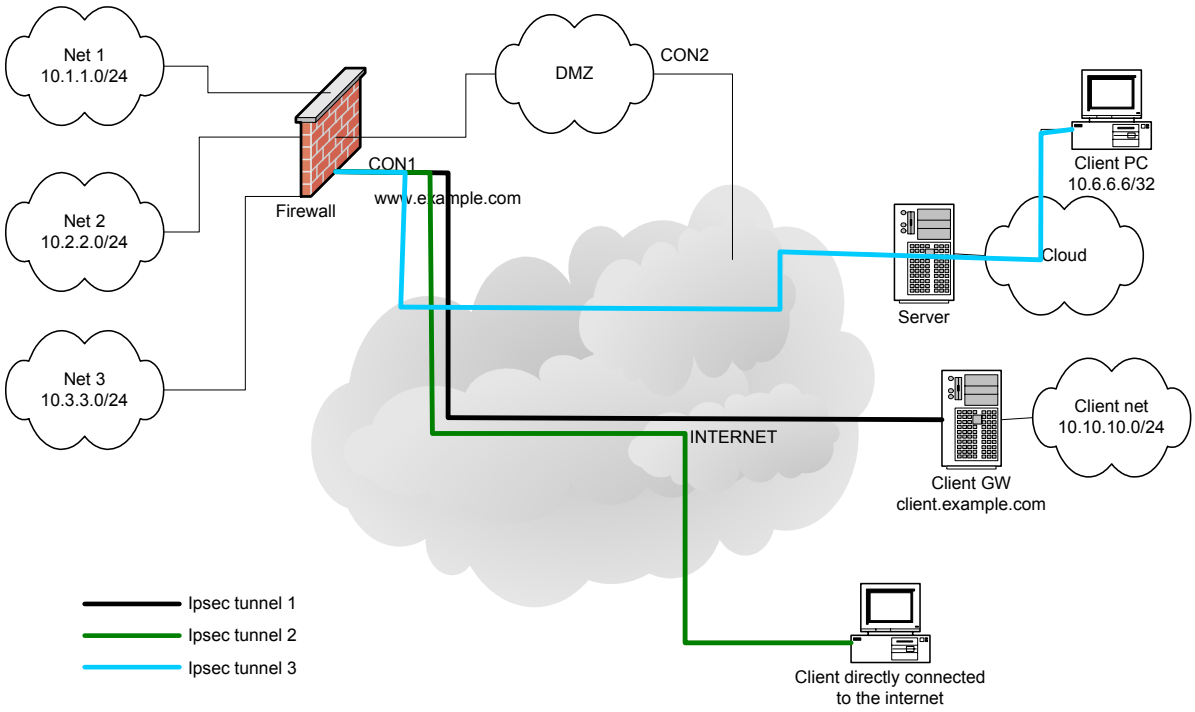
# Introduction

People want to, for more than one reason, work at home, using information from the corporate network. Companies want to link their networks together over the Internet, but nobody wants his or her network traffic to be readable on the Internet. Various solutions have been developed, and one of them is IPsec.

This paper covers the use of IPsec on a FreeBSD client and an OpenBSD server, both using isakmpd in an example environment. This environment is pretty generic, so what is used here can be easily translated to other organizations.

# Scenario

In the case described in this paper we have the following scenario:




The client must be able to connect from everywhere on the Internet independent from IP-addresses of the client. It might be connected via cable, which changes IP-address once in a while, or on a network behind such a connection. It can also be a gateway with a network behind it, let's say 10.10.10.0/24.

Since it does not have a fixed IP-address (as seen from the Internet) we call this a roaming client. In this paper it will be a FreeBSD machine running isakmpd and ipfilter.
The remote location can be configured as the roaming client.

The firewall has multiple interfaces. The client will enter the network on CON1. It is running OpenBSD with isakmpd and packet filter.

# IPsec / ISAKMP

## Introduction

IPsec is an open set of protocols, implemented in several operating systems by several companies. It does encryption, decryption and verification. It is implemented in the IP layer.

There are to protocols in IPsec:

- Authentication Header (AH) guarantees integrity of IP packet and protects it from intermediate alteration or impersonation. This is done by attaching a cryptographic checksum. This is protocol 51.
- Encapsulated Security Payload (ESP) protects the payload of an IP-packet by encrypting it with a secret key. This is protocol 51.

(Note that we are talking about protocols here, TCP is protocol 6, UDP is protocol 17)

The roaming client must be able to travel through a NATting firewall. Since NAT requires header rewriting it makes the AH-protocol unusable in this situation. So only ESP is used.

There are 2 modes to use IPsec in:

- Transport mode is used for peer-to-peer communication between nodes.
- Tunnel mode is used to let security gateways communicate with each other. Behind one or both gateways there can be a network using the tunnel-mode to communicate with the other end.

We use tunnel mode since the client has to reach a network behind the firewall.

Setting up an IPsec connection between two peers takes to phases:

- In the first phase, a secure channel is created to communicate between the peers. In this stage things like passphrase or certificates are exchanged.
- In the second phase, security associations (SA's) are negotiated by services such as IPsec. In this phase the details of the IPsec-link will be negotiated, such as encapsulation mode, lifetime and algorithms to use. Also the identities like host and network are specified.

Once these two steps are successfully finished, an encrypted tunnel exists. Later in this document, after showing how configuration and certificates are made, there is output of packets written by isakmpd ( -l <file> option) and decoded by tethereal.

## Security Policies and associations

*Policies*
During negotiating a security association (SA) and a security policy (SP) are made. The SP is ment to check wether an inbound or outbout IP packet applies to a SA.
Packets are identified by:

- Security Parameter Index (SPI)
- Source / Destination address
- ESP or AH protocol

For example, we have the following SP on the client from IPsec tunnel 3:
(192.0.34.72 is the firewall, 212.203.6.138 is the client, and 10.1.1.0/24 is the network behind the firewall.)

```
10.1.1.0/24[any] 212.203.6.138[any] any
       in ipsec
       esp/tunnel/192.0.34.72-212.203.6.138/use
       spid=26 seq=1 pid=27305
       refcnt=1
212.203.6.138[any] 10.1.1.0/24[any] any
       out ipsec
       esp/tunnel/212.203.6.138-192.0.34.72/require
       spid=25 seq=0 pid=27305
       refcnt=1
```

If a packet enters the client with destination 212.203.6.138, and it originates (encrypted) from 192.0.34.72 it will be decrypted according to what is associated in the SA (see below). If in the decrypted packet 10.1.1.0/24 is the source, the packet will pass, otherwise it will be dropped.

If a packet wants to leave the client, it is checked against the SP. If it has 10.1.1.0/24 as destination, from 212.203.6.138, it must be encrypted and sent to 192.0.34.72.

This information can be retrieved using the command 'setkey -DP' on FreeBSD.
On the OpenBSD server this information is shown with the command 'netstat':

```
$ netstat -rn -f encap
Routing tables

Encap:
Source             Port  Destination        Port  Proto
SA(Address/Proto/Type/Direction)
212.203.6.138/32   0     10.1.1.0/24          0     0     192.0.34.72/50/use/in
10.1.1.0/24        0     212.203.6.138/32     0     0
192.0.34.72/50/require/out
```

Packets that apply to one of those routing rules are treated accordingly.

*Associations*

If a packet passes the SP it is brought to the SA.
How the packets must be encrypted or decrypted is stored in the Security Associations Database (SAD). The content of this database can, on FreeBSD, be retreived using the command '*setkey -D*'. The content looks like this:

```
212.203.6.138 192.0.34.72
       esp mode=any spi=903662016(0x35dcc9c0) reqid=0(0x00000000)
       E: 3des-cbc  946b0af0 c3c001f7 87e87f4e 25eb9c73 e9a04dbe c1b8cfe1
       A: hmac-md5  080a57ad 6a82512c 0e40e506 f00a0578
       seq=0x00000000 replay=0 flags=0x00000000 state=mature
       created: Oct 11 11:55:40 2002    current: Oct 11 11:57:50 2002
       diff: 130(s)  hard: 600(s)  soft: 540(s)
       last:                       hard: 0(s)    soft: 0(s)
       current: 0(bytes)   hard: 0(bytes)      soft: 0(bytes)
       allocated: 0 hard: 0      soft: 0
       sadb_seq=1 pid=27304 refcnt=1
192.0.34.72 212.203.6.138
       esp mode=any spi=914016611(0x367ac963) reqid=0(0x00000000)
       E: 3des-cbc  89e40f1d 663e82e9 9130eb12 c48482d1 3dfa4c2b c05e1d40
       A: hmac-md5  ba89e8a3 fc523573 228df1d4 c46069b2
       seq=0x00000000 replay=0 flags=0x00000000 state=mature
       created: Oct 11 11:55:40 2002    current: Oct 11 11:57:50 2002
       diff: 130(s)  hard: 600(s)  soft: 540(s)
       last:                       hard: 0(s)    soft: 0(s)
       current: 0(bytes)   hard: 0(bytes)      soft: 0(bytes)
       allocated: 0 hard: 0      soft: 0
       sadb_seq=0 pid=27304 refcnt=1
```

## Configuration

*Kernel configuration*

We will now discuss how the server and the clients are configured. As example we take the IPsec connection #1, a gateway with a network 10.10.10.0/24 behind it, connecting to the firewall/VPN gateway to reach network 1.

First we need to make the kernel of the machines IPsec enabled.
On OpenBSD this is done by adding:

```
Option              IPSEC        # IPsec
pseudo-device       enc    1     # IPSEC needs the encapsulation interface
```

to the kernel configuration and make and install a kernel with this new configuration.

FreeBSD uses the lines:

```
Options      IPSEC         #IP security
Options      IPSEC_ESP     #IP security (crypto; define w/ IPSEC)
Options      IPSEC_DEBUG   #debug for IP security
```

Also for FreeBSD a new kernel must be built and installed.

Now the machines have an IPsec enabled kernel, the next step will be setting up IPsec. Before doing so, decisions have to be made on the type of authentication to use. Either passphrases or X509 certificates can be used. Since passphrases will be to weak for this environment, X509 certificates are used. This implies the need of a Certificate Authority (CA). You can go out on the Internet and go to a CA that can deliver you the needed certificates. But if you trust yourself and your network, you can setup your own.

*Setting up a Certificate Authority*

In our environment we have setup a dedicated machine that can only be accesses by certain people using ssh-keys. All other access to the machine is denied. This is the machine where certificates are created and stored. They must be in a safe place (we allow physical access to the CA-server, though...).

To be able to create a CA openssl is needed. Be sure to use one of the latest versions.

As root, you need to do the following:

```
# mkdir -p /etc/ssl/private
# openssl genrsa -out /etc/ssl/private/ca.key 1024
# openssl req -new -key /etc/ssl/private/ca.key \
        -out /etc/ssl/private/ca.csr
```

During the third step some questions needs to be answered. The answers are printed bold.

```
Country Name (2 letter code) [AU]:NL
State or Province Name (full name) [Some-State]:Groningen
Locality Name (eg, city) []:Groningen
Organization Name (eg, company) [Internet Widgits Pty Ltd]:EuroBSDcon
Organizational Unit Name (eg, section) []:EuroBSDcon CA dept.
Common Name (eg, YOUR name) []:BOFH
Email Address []:bofh@eurobsdcon.org

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:Tru5t'nmyCA
An optional company name []:
```

Now we have a CA key (ca.key) and a certificate signing request (ca.csr) file. The latter one is used to create a certificate that will be signed with the key:

```
# openssl x509 -req -days 365 -in /etc/ssl/private/ca.csr \
        -signkey /etc/ssl/private/ca.key \
        -extfile /etc/ssl/x509v3.cnf -extensions x509v3_CA \
        -out /etc/ssl/ca.crt
```

Note that the file "/etc/ssl/x509v3.cnf" is not available in a default FreeBSD install. However it can be fetched from OpenBSD and used without problems.

Now we have a CA certificate, ca.crt. Together with the CA key this certificate is used to create client certificates. Setting up a CA is done.

*Creating client certificates*

Creating certificates is done on the same machine as the CA because the keys of the CA are needed. For each peer, certificates are needed. Since the solution needs to be IP-address independent, we choose to use FQDN certificates.

To store the certificates for more than one peer, a directory structure is set up like it would be used in isakmpd:

```
/usr/local/cert/FQDN1/ca
              |    |--/certs
              |    |--/private
              |
              |-/FQDN2/ca
              |    |--/certs
              |    |--/private
              |
```

where 'FQDN{1|2}' represents the FQDN of the peer, e.g. www.example.com. For each new peer a new set of directories is created.

After creating this directory structure a certificate signing request for the server is created:

```
# openssl genrsa -out
/usr/local/cert/www.example.com/private/www.example.com.key 1024
# openssl req -new -key
/usr/local/cert/www.example.com/private/www.example.com.key \
              -out /usr/local/cert/www.example.com/private/www.example.com.csr
```

Now the certificate itself must be created for the server:

```
# cd /usr/local/cert/www.example.com/private
# setenv CERTFQDN www.example.com
# openssl x509 -req -days 365 -in www.example.com.csr \
        -CA /etc/ssl/ca.crt -CAkey /etc/ssl/private/ca.key \
        -CAcreateserial \
        -extfile /etc/ssl/x509v3.cnf -extensions x509v3_FQDN \
        -out www.example.com.crt
```

Now the certificate is copied to the 'cert' directory and the CA cert is included:

```
# cp www.example.com.crt ../certs/
# cp /etc/ssl/ca.crt /usr/local/cert/www.example.com/ca
```

Now there is a complete set of certificates that can be transferred to the server.
For the client the same steps must be performed, but now with the FQDN the client will use. This can be a non-existing FQDN like 'client.example.com' since the mechanism of checking certificates only checks an abstract of the certificate and it is not checked against the real FQDN. This enables the use for roaming clients.

*Distribution and use of certificates*

At this stage the isakmpd configuration files are entering the scene.
On the server side, a generic setup is needed which will allow all clients to connect and authenticate, without having the configuration file updated each time a client is added. The same CA signs certificates from the client and server. When peers authenticate each other they will create an abstract of information from the received certificate and compare that with an abstract of what is in the policy file. Since the abstract of the CA certificate will be the same, it is possible to add the ca.crt to the policy file and distribute this to all the peers (including the server). In this case the policy file needs to be edited (depending on some settings).

We can create a generic policy file that will look like this:

```
# cat isakmpd.policy
Comment: This policy accepts ESP SAs from a remote that uses the right
password
        $OpenBSD: policy,v 1.6 2001/06/20 16:36:19 angelos Exp $
        $EOM: policy,v 1.6 2000/10/09 22:08:30 angelos Exp $
Authorizer: "POLICY"
licensees: "x509-base64:\
        MIIC1TCCAj6gAwIBAgIBADANBgkqhkiG9w0BAQQFADCBnTELMAkGA1UEBhMCTkwx\
        EjAQBgNVBAgTCUdyb25pbmdlbjESMBAGA1UEBxMJR3JvbmluZ2VuMRUwEwYDVQQK\
        EwxCcmFzYXBlbiBvcmcxFzAVBgNVBAsTDkhlYWRxdWFydGVyIENBMRIwEAYDVQQD\
        EwlFaWxrbyBCb3MxIjAgBgkqhkiG9w0BCQEWE3RhZmthbUBicmFzYXBlbi5vcmcw\
        HhcNMDIU                                                EBhMCTkwx\
        EjAQBgNV   THIS IS THE ca.crt from the ./ca directory  wEwYDVQQK\
        EwxCcmFz                                               wEAYDVQQD\
        EwlFaWxrbyBCb3MxIjAgBgkqhkiG9w0BCQEWE3RhZmthbUBicmFzYXBlbi5vcmcw\
        gZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJjcx3Hg66XEDz9kDX+6qE5lgX6e\
        70kbrmV1CQXyYV2wc3p9+bUjqqkm5AVYj3bUfyXOrngSSnPNQBO0Jqv8iRWcoGvy\
        qBeqYvxvmyoysTt5FAregstvg34whgbh375u65yh45egbdfvbtzNHq1nl7z+DRF9\
        UN88scqffYm/AJrPAgMBAAGjIzAhMBIGA1UdEwEB/wQIMAYBAf8CAQEwCwYDVR0P\
        BAQDAgKEMA0GCSqGSIb3DQEBBAUAA4GBAFdYPJEz5keXA7/6/JE4W5pFWSjnsPj3\
        wC/qR+ki/dQ0tAgt15f9W6zI6BUs8xUP4Va4EppbdDxIhyX+aiMJ3ETMlMu2h/rZ\
        MaBABByPZzNmMwP1b5J8rEP4rk9VwHpbQFA0RFJdCDo2ekIrcVM2bT89cz5IS9lGT\
        NWLa0vTsNm9b"
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg == "aes" &&
            esp_auth_alg == "hmac-sha" -> "true";
```

As extra check the Conditions can be added with the FQDN of the client:

```
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg == "aes" &&
            esp_auth_alg == "hmac-sha" &&
            (remote_id == "client.example.com"  ||
             remote_id == "another.example.com") -> "true";
```

This policy can be copied to the /usr/local/cert/FQDN directory for each client, so it can be transferred together with the certificates.

*ISAKMPD configuration files*

How a policy file is created is described in the previous part.
The only thing missing is a configuration file. This one differs from peer to peer, but a template can be generated. Let's walk though the configuration file of the server first. this file is different from the rest. Since it must be possible to use roaming users no IP-addresses of the client must appear in the configuration file. Furthermore, to improve readability, all unneeded lines are removed.

It looks like this, lines starting with a '#' and italic font are comments of the author.

```
# $ Id: $
# Paths in this file are paths as they would be on
# the final destination, in this case the server.
# General section
[General]
Policy-File=
/usr/local/etc/isakmpd/policy
Retransmits=            5
Exchange-max-time=      120
Listen-on=              192.0.34.72

# Incoming phase 1 negotiations are multiplexed on
# the source IP address
[Phase 1]
Default=                ISAKMP-peer-GNU

# These connections are walked over after config
# file parsing and told to the application layer
# so that it will inform us when traffic wants to
# pass over them. This means we can do on-demand
# keying.
[Phase 2]
Connections=            IPsec-OBSD-GNU

# The peers
[ISAKMP-peer-GNU]
Phase=                  1
Transport=              udp
Local-address=          192.0.34.72
# We don't want the peer in the configfile.
# Instead we point to our identity and leave the
# rest to certificates and the policy file
ID=                     work-ID
Configuration=          Default-main-mode

[work-ID]
ID-type=                FQDN
Name=                   www.example.com

# The different connections
[IPsec-OBSD-GNU]
Phase=                  2
ISAKMP-peer=            ISAKMP-peer-GNU
Configuration=          Default-quick-mode
Local-ID=               Net-OBSD
Remote-ID=              Net-GNU

# Our Networks
[Net-GNU]
# This is the remote network. We leave it generic,
# it will be negotiated, and the configuration of
# the peer will be used.
ID-type=                IPV4_ADDR_SUBNET
Network=                0.0.0.0
Netmask=                0.0.0.0

[Net-OBSD]
ID-type=                IPV4_ADDR_SUBNET
Network=                10.1.1.0
Netmask=                255.255.255.0

# Certificates stored in PEM format
[X509-certificates]
CA-directory=           /usr/local/etc/isakmpd/ca/
Cert-directory=
/usr/local/etc/isakmpd/certs/
Private-key=     \
/usr/local/etc/isakmpd/private/www.example.com.key

# Phase 1 descriptions
[Default-main-mode]
DOI=                    IPSEC
EXCHANGE_TYPE=          ID_PROT
Transforms=             3DES-SHA,3DES-MD5

# Main mode transforms
########################
# 3DES
# Note "AUTHENTICATION_METHOD" it is not
#"PRE_SHARED"
# as it would by default
[3DES-SHA]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         SHA
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                   LIFE_3600_SECS

[3DES-MD5]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         MD5
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                   LIFE_3600_SECS

< snip rest of default config file >
```

This file can be stored in the directory /usr/local/certs/www.example.com/.
Now the files for the server are all in place. They can be transferred to the server in /usr/local/etc/isakmpd(/{ca|certs|private}/). Change the mode of the files to 600 so others cannot read them. If this is not done, isakmpd refuses to read the configuration file (*too open permissions*) and will not work.

What is left is the configuration of the client. The configuration file looks pretty much the same but in this file the peer (server) is explicitly named:

```
# cat /usr/local/certs/isakmpd.conf.template
```

```
# General section
# The IP-address of the client is
# @@MY_IP_ADDRESS@@ so it can be
substituted.
# The FQDN is @@MY_FQDN@@
[General]
Retransmits=            5
Exchange-max-time=      120
Listen-on=
@@MY_IP_ADDRESS@@
# Listen-on=            212.203.6.138

# Incoming phase 1 negotiations are
multiplexed on the source IP address
[Phase 1]
193.78.174.81=          ISAKMP-peer-GNU
Default=                ISAKMP-peer-GNU

# These connections are walked over
after config
# file parsing and told to the
application layer
# so that it will inform us when traffic
wants
# to pass over them. This means we can
do on-
# demand keying.
[Phase 2]
Connections=            IPsec-OBSD-GNU

# The peers
[ISAKMP-peer-GNU]
# over here we explicitely declare the
server's
# address.
Phase=                  1
Transport=              udp
Local-address=
@@MY_IP_ADDRESS@@
Address=                192.0.34.72
ID=                     my-ID
Configuration=          Default-main-
mode

[my-ID]
ID-type=                FQDN
Name=                   @@MY_FQDN@@
# Name=
client.example.com

# The different connections
[IPsec-OBSD-GNU]
Phase=                  2
ISAKMP-peer=            ISAKMP-peer-GNU
Configuration=          Default-quick-
mode
Local-ID=               Net-OBSD
Remote-ID=              Net-GNU
```

```
# Our Networks
[Net-GNU]
# this is the remote network to be reached.
ID-type=                IPV4_ADDR_SUBNET
Network=                10.1.1.0
Netmask=                255.255.255.0

[Net-OBSD]
# this is our network
ID-type=                IPV4_ADDR_SUBNET
Network=                10.10.10.0
Netmask=                255.255.255.0

# Certificates stored in PEM format
[X509-certificates]
CA-directory=    /etc/isakmpd/ca/
Cert-directory=  /etc/isakmpd/certs/
Private-key= \
/etc/isakmpd/private/@@MY_FQDN@@.key

# Phase 1 descriptions
[Default-main-mode]
DOI=                    IPSEC
EXCHANGE_TYPE=          ID_PROT
Transforms=             3DES-SHA,3DES-MD5

# Main mode transforms
########################
# 3DES

[3DES-SHA]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         SHA
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                   LIFE_3600_SECS

[3DES-MD5]
ENCRYPTION_ALGORITHM=   3DES_CBC
HASH_ALGORITHM=         MD5
AUTHENTICATION_METHOD=  RSA_SIG
GROUP_DESCRIPTION=      MODP_1024
Life=                   LIFE_3600_SECS

< snip rest of default configfile >
```

This file can be used for every new client, only the @@MY_IP_ADDRESS@@ and @@MY_FQDN@@ must be substituted. Copy this template to /usr/local/certs/FQDN(client)/ and do the substitution.

Now the configuration of the client is done as well. The directory /usr/local/certs/FQDN(client) can be transferred to the client machine in /usr/local/etc/isakmpd/.

## Setting up the connection

Now on both sides isakmpd can be started. For debugging purposes both can write everything to STDOUT (-D A=99), which is very noisy. It is better to redirect it to a file. Also both can be started with the '-l <file>' option, causing isakmpd to write negotiation packets to a pcap-file.

On both sides isakmpd is (for debugging) started as:

```
# cd /usr/local/etc/isakmpd
# isakmpd -c isakmpd.conf -l /tmp/ike.pcap -D A=99 -d > /tmp/ike.debug 2>&1 &
```

The file /tmp/ike.pcap can now be read into tethereal (which is installed with the net/ethereal port):

```
# tethereal -V -r /tmp/ike.pcap > /tmp/ike_readable.txt
```

The result is a file that is human readable. The negotiation took 9 packets in total:

```
22:20:02.015224 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 1 I ident: [|sa]
22:20:02.160553 212.203.6.138.500 > 192.0.34.72.500: isakmp: phase 1 R ident: [|sa]
22:20:02.242995 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 1 I ident: [|ke]
22:20:02.516224 212.203.6.138.500 > 192.0.34.72.500: isakmp: phase 1 R ident: [|ke]
22:20:02.676648 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 1 I ident[E]:
       [encrypted id]
22:20:03.478109 212.203.6.138.500 > 192.0.34.72.500: isakmp: phase 1 R ident[E]:
       [encrypted id]
22:20:03.590456 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 2/others I
       oakley-quick[E]: [encrypted hash]
22:20:03.963916 212.203.6.138.500 > 192.0.34.72.500: isakmp: phase 2/others R
       oakley-quick[E]: [encrypted hash]
22:20:03.986819 192.0.34.72.500 > 212.203.6.138.500: isakmp: phase 2/others I
       oakley-quick[E]: [encrypted hash]
```

Below is the (snipped) output of each packet from the servers side, the connection is iniated by the server:

```
Frame 1 (144 on wire, 144 captured)
Null/Loopback
    Family: IP (0x00000002)
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
    Version: 4
    Protocol: UDP (0x11)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Initiator cookie, Responder cookie
    Next payload: Security Association (1)
    Version: 1.0
    Exchange type: Identity Protection (Main Mode) (2)
    Flags
        .... ...0 = No encryption
        .... ..0. = No commit
        .... .0.. = No authentication
    Message ID: 0x00000000
    Length: 112
    Security Association payload
        Next payload: NONE (0)
        Length: 84
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload
            Next payload: NONE (0)
            Length: 72
```

```
            Proposal number: 1
            Protocol ID: ISAKMP (1)
            SPI size: 0
            Number of transforms: 2
            Transform payload
                Next payload: Transform (3)
                Length: 32
                Transform number: 0
                Transform ID: KEY_IKE (1)
                Encryption-Algorithm (1): 3DES-CBC (5)
                Hash-Algorithm (2): SHA (2)
                Authentication-Method (3): RSA-SIG (3)
                Group-Description (4): Group-Value (2)
                Life-Type (11): Seconds (1)
                Life-Duration (12): Duration-Value (3600)
            Transform payload
                Next payload: NONE (0)
                Length: 32
                Transform number: 1
                Transform ID: KEY_IKE (1)
                Encryption-Algorithm (1): 3DES-CBC (5)
                Hash-Algorithm (2): MD5 (1)
                Authentication-Method (3): RSA-SIG (3)
                Group-Description (4): Group-Value (2)
                Life-Type (11): Seconds (1)
                Life-Duration (12): Duration-Value (3600)

Frame 2 (112 on wire, 112 captured)
Null/Loopback
Internet Protocol, Src Addr: client.example.com (212.203.6.138), Dst Addr:
www.example.com (192.0.34.72)
    Version: 4
    Protocol: UDP (0x11)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Security Association (1)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 80
    Security Association payload
        Next payload: NONE (0)
        Length: 52
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload
            Next payload: NONE (0)
            Length: 40
            Proposal number: 1
            Protocol ID: ISAKMP (1)
            SPI size: 0
            Number of transforms: 1
            Transform payload
                Next payload: NONE (0)
                Length: 32
                Transform number: 0
                Transform ID: KEY_IKE (1)
                Encryption-Algorithm (1): 3DES-CBC (5)
                Hash-Algorithm (2): SHA (2)
                Authentication-Method (3): RSA-SIG (3)
                Group-Description (4): Group-Value (2)
                Life-Type (11): Seconds (1)
                Life-Duration (12): Duration-Value (3600)
```

```
Frame 3 (212 on wire, 212 captured)
Null/Loopback
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Key Exchange (4)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 180
    Key Exchange payload
        Next payload: Nonce (10)
        Length: 132
        Key Exchange Data
    Nonce payload
        Next payload: NONE (0)
        Length: 20
        Nonce Data

Frame 4 (212 on wire, 212 captured)
Null/Loopback
Internet Protocol, Src Addr: client.example.com (212.203.6.138), Dst Addr:
www.example.com (192.0.34.72)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Key Exchange (4)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 180
    Key Exchange payload
        Next payload: Nonce (10)
        Length: 132
        Key Exchange Data
    Nonce payload
        Next payload: NONE (0)
        Length: 20
        Nonce Data

Frame 5 (999 on wire, 999 captured)
Null/Loopback
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Identification (5)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 967
    Identification payload
        Next payload: Certificate (6)
        Length: 32
        ID type: FQDN (2)
        Protocol ID: Unused
        Port: Unused
        Identification data: www.example.com
    Certificate payload
        Next payload: Signature (9)
        Length: 747
        Certificate encoding: 4 - X.509 Certificate - Signature
        Certificate Data
    Signature payload
        Next payload: Notification (11)
        Length: 132
        Signature Data
    Notification payload
        Next payload: NONE (0)
        Length: 28
```

```
        Domain of Interpretation: IPSEC (1)
        Protocol ID: ISAKMP (1)
        SPI size: 16
        Message type: INITIAL-CONTACT (24578)
        Security Parameter Index

Frame 6 (980 on wire, 980 captured)
Null/Loopback
Internet Protocol, Src Addr: client.example.com (212.203.6.138), Dst Addr:
www.example.com (192.0.34.72)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Identification (5)
    Exchange type: Identity Protection (Main Mode) (2)
    Message ID: 0x00000000
    Length: 948
    Identification payload
        Next payload: Certificate (6)
        Length: 24
        ID type: FQDN (2)
        Protocol ID: Unused
        Port: Unused
        Identification data: client.example.com
    Certificate payload
        Next payload: Signature (9)
        Length: 732
        Certificate encoding: 4 - X.509 Certificate - Signature
        Certificate Data
    Signature payload
        Next payload: Notification (11)
        Length: 132
        Signature Data
    Notification payload
        Next payload: NONE (0)
        Length: 28
        Domain of Interpretation: IPSEC (1)
        Protocol ID: ISAKMP (1)
        SPI size: 16
        Message type: INITIAL-CONTACT (24578)
        Security Parameter Index
    Extra data: 00000000

Frame 7 (320 on wire, 320 captured)
Null/Loopback
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Hash (8)
    Exchange type: Quick Mode (32)
    Message ID: 0x29a5c670
    Length: 288
    Hash payload
        Next payload: Security Association (1)
        Length: 24
        Hash Data
    Security Association payload
        Next payload: Nonce (10)
        Length: 52
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload
            Next payload: NONE (0)
            Length: 40
            Proposal number: 1
            Protocol ID: IPSEC_ESP (3)
            SPI size: 4
```

```
            Number of transforms: 1
            SPI: 4FF93BE9
            Transform payload
                Next payload: NONE (0)
                Length: 28
                Transform number: 1
                Transform ID: 3DES (3)
                SA-Life-Type (1): Seconds (1)
                SA-Life-Duration (2): Duration-Value (600)
                Encapsulation-Mode (4): Tunnel (1)
                Authentication-Algorithm (5): HMAC-MD5 (1)
                Group-Description (3): Group-Value (2)
    Nonce payload
        Next payload: Key Exchange (4)
        Length: 20
        Nonce Data
    Key Exchange payload
        Next payload: Identification (5)
        Length: 132
        Key Exchange Data
    Identification payload
        Next payload: Identification (5)
        Length: 16
        ID type: IPV4_ADDR_SUBNET (4)
        Protocol ID: Unused
        Port: Unused
        Identification data: 10.1.1.0/255.255.255.0
    Identification payload
        Next payload: NONE (0)
        Length: 16
        ID type: IPV4_ADDR_SUBNET (4)
        Protocol ID: Unused
        Port: Unused
        Identification data: 212.203.6.138/255.255.255.255

Frame 8 (324 on wire, 324 captured)
Null/Loopback
Internet Protocol, Src Addr: client.example.com (212.203.6.138), Dst Addr:
www.example.com (192.0.34.72)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Hash (8)
    Exchange type: Quick Mode (32)
    Message ID: 0x29a5c670
    Length: 292
    Hash payload
        Next payload: Security Association (1)
        Length: 24
        Hash Data
    Security Association payload
        Next payload: Nonce (10)
        Length: 52
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload
            Next payload: NONE (0)
            Length: 40
            Proposal number: 1
            Protocol ID: IPSEC_ESP (3)
            SPI size: 4
            Number of transforms: 1
            SPI: 5838A401
            Transform payload
                Next payload: NONE (0)
                Length: 28
                Transform number: 1
                Transform ID: 3DES (3)
```

```
                SA-Life-Type (1): Seconds (1)
                SA-Life-Duration (2): Duration-Value (600)
                Encapsulation-Mode (4): Tunnel (1)
                Authentication-Algorithm (5): HMAC-MD5 (1)
                Group-Description (3): Group-Value (2)
    Nonce payload
        Next payload: Key Exchange (4)
        Length: 20
        Nonce Data
    Key Exchange payload
        Next payload: Identification (5)
        Length: 132
        Key Exchange Data
    Identification payload
        Next payload: Identification (5)
        Length: 16
        ID type: IPV4_ADDR_SUBNET (4)
        Protocol ID: Unused
        Port: Unused
        Identification data: 10.1.1.0/255.255.255.0
    Identification payload
        Next payload: NONE (0)
        Length: 16
        ID type: IPV4_ADDR_SUBNET (4)
        Protocol ID: Unused
        Port: Unused
        Identification data: 212.203.6.138/255.255.255.255
    Extra data: 00000000

Frame 9 (84 on wire, 84 captured)
Null/Loopback
Internet Protocol, Src Addr: www.example.com (192.0.34.72), Dst Addr:
client.example.com (212.203.6.138)
User Datagram Protocol, Src Port: isakmp (500), Dst Port: isakmp (500)
Internet Security Association and Key Management Protocol
    Next payload: Hash (8)
    Exchange type: Quick Mode (32)
    Message ID: 0x29a5c670
    Length: 52
    Hash payload
        Next payload: NONE (0)
        Length: 24
        Hash Data
```

The message ID of the phase 1 part of the conversation is 0x00000000. As soon as phase 2 is reached, the message ID is changed to 0x29a5c670. In this conversation it is clear to see that in phase 1 authentication is done, while negotiation about network properties is done in phase 2.

## Helpful tools

This conversation resulted in a correct working Ipsec link. But in many cases of new developed IPsec links, many things can go wrong. Captured packets as shown before are nice to have for debugging but will often not suffice. Looking at the outputfile of isakmpd can help a lot. As said before, this output is very noisy, but it contains a lot of useful information.

It can happen that routing tables are not correctly setup. E.g. a VPN-client with 2 IP-addresses on the same interface might place packets destined for the tunnel on the wrong IP-address. Look at the routing table to see if something missing.

In the following chapter a brief overview of firewall settings will be given. When testing IPSec in secure environments, firewalls might cause problems with respect to negotiation or ESP-traffic. If possible, stop the firewall for testing, or log all the blocked traffic. On FreeBSD this is done with ‘ipmon(8)’. On OpenBSD this is done with a tcpdump on pflog (man pflog).

# Firewalling

## Client site

There are some demands on the client site for firewalling. It is not allowed to access the Internet directly. If an IPsec connection is active, all Internet activity should go via the proxy server on network 1. DNS-requests are sent to the name server on network 1.

To keep control over the traffic, only a few protocols are allowed to work remotely. These protocols are ICA for Citrix and ssh.

Assuming that the network interface to the Internet is called 'fxp0' and the internal interface called 'fxp1', a firewall configuration on the client gateway running FreeBSD with ipfilter, would look like this:

```
# 10.10.10.1 = internal address of client gateway
# 10.10.10.2 = nameserver
# 10.10.10.16 = admin's PC
# No antispoofing is needed on this interface. We only allow from one IP
address.
# Allow incoming and outgoing IPsec traffic from the server
pass in quick on fxp0 proto udp from 97.0.34.72 to 212.203.6.138 port = 500 \
   keep state
pass out quick on fxp0 proto udp from 212.203.6.138 to 97.0.34.72 port = 500 \
   keep state
pass in quick on fxp0 proto esp from 97.0.34.72 to 212.203.6.138
pass out quick on fxp0 proto esp from 212.203.6.138 to 97.0.34.72
pass in quick on lo0 all
pass out quick on lo0 all
# Allow admin access from admin's PC
pass in quick on fxp1 proto tcp from 10.10.10.16 to 10.10.10.1 port = ssh \
   keep state
# allow outgoing dns queries
pass out quick on fxp1 proto udp from 10.10.10.1 port=53 to 10.10.10.2 \
   port=52 keep state
block in log quick on fxp1 from any to 10.10.10.1
# Perform some security on incoming traffic. It must come from network 1 or
# ourselves.
pass in quick on fxp1 from 10.10.10.0/24 to 10.1.1.0/24
pass in quick on fxp1 from 10.10.10.0/25 to 10.10.10.0/24
pass out quick on fxp1 from 10.1.1.0/24 to 10.10.10.0/24
pass out quick on fxp1 from 10.10.10.0/24 to 10.10.10.0/24
# block and log the rest
block in log level auth.alert quick all
block out log level auth.alert quick all
```

Note that this very simple firewall configuration does not allow e.g. ICMP traffic. This might cause problems with e.g. MTU discovery or other administrative network information. Restriction on type of traffic will be done on the server, since firewalling on the client is not that reliable. If it happens to get compromised it is easy to change the rules.

## Server site

This site is a bit more complicated. On this site traffic of the client is checked, but that can only be done after it is decrypted. So it can't be done on the incoming interface. We filter on other interfaces.
The server is running OpenBSD with packet filter. It has 5 interfaces. For VPN traffic only 2 interfaces are used: the interface to the Internet where the VPN-traffic comes in and the interface to network 1. It is easy to add rules so VPN-traffic can go to the other networks as well.

```
# See pf.conf(5) for syntax and examples
# interfaces by network
lo_if         = "lo0"
ext_if        = "rl0"
net1_if       = "dc0"
net2_if       = "dc1"
net3_if       = "dc2"
dmz_if        = "dc3"
# network addresses
ifext_net     = "rl0/8"
if1_net       = "dc0/24"
if2_net       = "dc1/24"
if3_net       = "dc2/24"
ifdmz_net     = "dc3/24"
internal_net  = "10.0.0.0/8"      # internal network, including VPN networks
ifext_ip      = "192.0.34.72"     # www.example.com
# special machines for which there are distinct rules
proxy         = "10.1.1.10
nameserver    = "10.1.1.11"
citrix        = "10.1.1.12"
caserver      = "10.1.1.14"
admin         = "10.1.1.100"

#############################################################################
#  The Rules
#############################################################################
# default deny
block in log all
block out log all

# loopback interace
pass in quick on $lo_if all
pass out quick on $lo_if all

# VPN/Internet interface
# rules for $ext_if (in)
pass in quick on $ext_if proto udp from any port = isakmp to $ext_if_ip \
   port = isakmp keep state
pass in quick on $ext_if proto esp from any to $ext_if_ip keep state

# other rules for $ext_if (out)
pass out quick on $ext_if proto udp from $ext_if_ip port = isakmp to any \
   port = isakmp keep state
pass out quick on $ext_if proto esp from $ext_if_ip to any keep state
pass out quick on $ext_if proto icmp from $ext_if_ip to any icmp-type echoreq
keep state
pass out quick on $ext_if proto tcp from $proxy to any keep state
pass out quick on $ext_if proto tcp from $nameserver port = 53 to any \
   port = 53 keep state
pass out quick on $ext_if proto udp from $nameserver to any port = 53 keep
state

pass in quick on enc0 all
pass out quick on enc0 all
```

```
####################
# network 1 interface
####################
# ICA traffic
pass out quick on $net1_if proto tcp from $internal_net to $citrix \
   port = 1496 keep state
pass out quick on $net1_if proto udp from $internal_net to $citrix \
   port = 1604 keep state
# DNS traffic
pass out quick on $net1_if proto tcp from $internal_net port= 53 \
   to $nameserver port = 53 keep state
pass out quick on $net1_if proto udp from $internal_net to $nameserver \
   port = 53 keep state
# Proxy traffic
pass out quick on $net1_if proto tcp from $internal_net to $proxy \
   port = 3128 keep state
# ssh traffic
block out quick on $net1_if proto tcp from $internal_net to $net1_if port = 22
pass out quick on $net1_if proto tcp from $internal_net to $if1_net port = 22
keep state

# Incoming for management
pass in quick on $net1_if proto tcp from {$caserver, $admin, $proxy} to \
   $net1_if port = ssh flags S/FSRA keep state
block in log quick on $net1_if from any to $net1_if
block out log quick on $net1_if from $net1_if to any
```

This is a very simple firewall setup,but it allows certain traffic from the VPN-client to some services on network 1. Note that only some services are allowed. Blocking the rest at this stage, after the tunnel, is more reliable than blocking it at the client side.

## Extra attention

The setup above applies to a client acting as a single machine behind a gateway or as a single machine directly connected to the Internet.

To avoid changing the configurations too much when the IP-address given by the ISP changes, it is possible to add an extra IP-address on the network interface:

```
# ifconfig ep0 add 10.7.7.7 netmask 255.255.255.0
# route add 10.0.0.0/8 10.7.7.7
```

An extra rule in the firewall configuration will be needed to allow traffic from 10.7.7.7 on the interface. After this has been done, the isakmpd configuration can be configured to listen on 10.7.7.7 instead of the external IP-address. When that external IP-address changes, only the firewall configuration needs to be changed and reloaded.