



**EarthLink™**

# IPv6 Prefix Delegation Using DHCPv6 Over IPSEC Tunnels

Research and Development  
EarthLink, Inc.  
June, 2005

Tim Bosserman  
Consulting Engineer

Copyright©2005  
EarthLink, Inc.

All Rights Reserved

Proprietary information of EarthLink, Inc. or its affiliates is contained therein. Any reproduction, use, appropriation, or disclosure of this information, in whole or in part, without the specific written authorization of the owner thereof is strictly prohibited. Failure to observe this notice may result in legal proceedings or liability for resulting damage or loss.

## 1 Introduction

EarthLink wishes to provide its customers with IPv6 connectivity. The motivation is to allow end-users to take advantage of the nearly limitless address space provided by IPv6. This would provide for true peer-to-peer networking applications without the limitations and functionality problems caused by NAT traversal issues. The goal was to devise a solution whereby:

1. Each end-user network would be delegated a /64 network prefix.
2. Such delegations would be authenticated.
3. Existing EarthLink authentication method(s) must be used.
4. No configuration of individual end-user computers would be required. All computers must auto-configure themselves.
5. Delegated prefixes would be static so individual computers would auto-configure themselves with static IP addresses.
6. No manual, per-customer configuration changes of datacenter routers would be needed.
7. No back-end data management of allocated IP space would be needed.

*Automatic 6to4 Tunneling* could be used to delegate IPv6 network prefixes to end users. Automatic 6to4 tunneling works well, but it has two significant drawbacks: the end-users must have a static IP address, and a static route must be configured into the router for each prefix to be delegated. This paper explores an alternative approach: DHCP version 6 (DHCPv6).

Many different tasks were undertaken during the course of this research and several new / unfamiliar technologies were explored:

1. DHCPv6 was used to dynamically delegate IPv6 prefixes.
2. IPv6-in-IP tunneling (IP protocol 41) was used to tunnel the DHCPv6 traffic from the end node to the Cisco Router.
3. IPSEC Header Authentication (AH) was used as a means of authenticating end nodes.
4. Internet Key Exchange (IKE) protocol was used for key establishment during IPSEC setup.
5. RADIUS authentication was used to provide the Cisco with the pre-shared keys during IKE Phase 1 Negotiation.

The following explains all of this in detail.

## 2 Delegating Prefixes via DHCPv6

One of the primary motivations in this research was to devise a way to delegate IPv6 prefixes without requiring any per-user configuration changes of datacenter equipment. DHCPv6 is the answer. It permits prefixes to be allocated to end users dynamically out of a pool configured on the Cisco. When a prefix gets allocated, a route is automatically added in the Cisco, routing the delegated prefix to the DHCPv6 client. When clients reboot or otherwise re-connect, they identify themselves via a DUID (DHCP Unique Identifier). The router will keep allocating the same prefix to the same DUID as long as it does not run out of available prefixes in its prefix pool. This is ideal, as it means if a customer cancels (or just stops using their prefix for a long time), the prefix will eventually get re-allocated to another customer.

The decision was made to break the work into phases:

1. Get DHCPv6 to work on a local LAN (the simplest possible case).
2. Get DHCPv6 to work over a tunnel to an end-user with broadband service.
3. Devise an authentication mechanism so prefix delegations may be limited to authorized users / customers.

### 2.1 DHCPv6 on local LAN

This was the first step in getting familiar with the workings of DHCPv6. First, the Cisco had to be configured to act as a DHCPv6 server. Second, a client machine had to be configured as a DHCPv6 client. The client machine also had to be configured as a router itself, since the purpose of prefix delegation is to route an entire IPv6 network to a router / gateway. Finally, a client machine was put behind the gateway in order to test for proper network connectivity and routing. Figure 1 illustrates.

#### 2.1.1 Configuring the Cisco Router

Configuring DHCPv6 on the Cisco is a two step process: first a “prefix pool” must be created, then DHCP is enabled on an interface and the interface is told which pool from which it is to allocate prefixes. Here are the exact configuration commands entered, along with a detailed description of each command:

```
ipv6 dhcp pool ipv6-pool1
prefix-delegation pool ipv6-pool1 lifetime 900 300
exit
ipv6 local pool ipv6-pool1 2001:470:115:C::/62 64

interface FastEthernet0/1
ipv6 dhcp server ipv6-pool1
exit
```

```
ipv6 dhcp pool ipv6-pool1
```

Creates a DHCPv6 pool and names it “ipv6-pool1”.

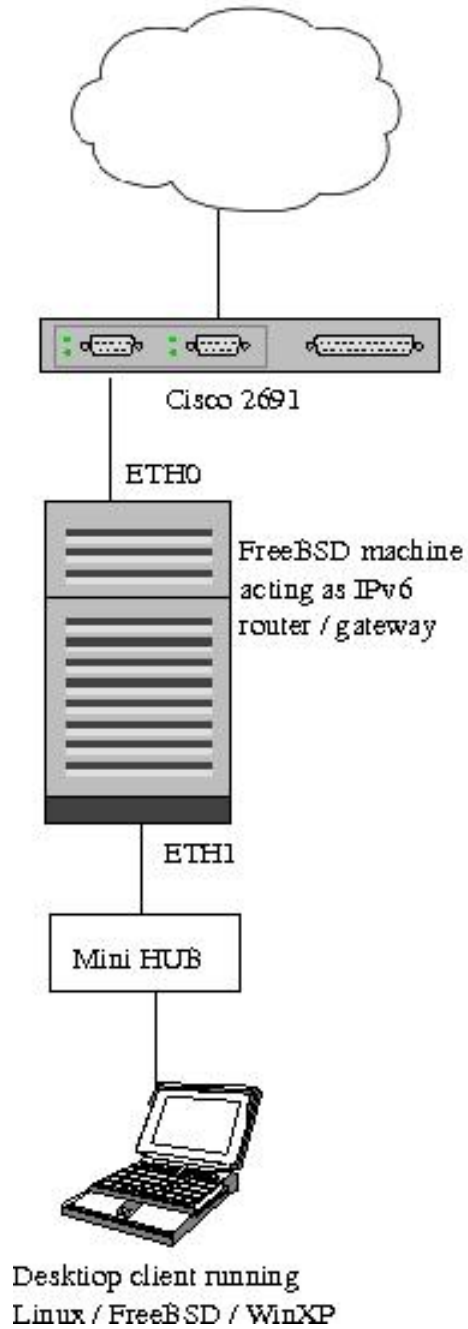


Figure 1: DHCPv6 over LAN

prefix-delegation pool ipv6-pool1 lifetime 900 300

Defines the attributes of ipv6-pool1. It is a prefix-delegation pool and it has a *valid lifetime* of 900 seconds and a *preferred lifetime* of 300 seconds. See below for more details on these numbers.

ipv6 local pool ipv6-pool1 2001:470:115:C::/62 64

Adds a network prefix pool to ipv6-pool1. In this case, the pool is being assigned a prefix of 2001:470:115:C::/62. The “64” at the end is the length of each prefix which is to be allocated upon a DHCPv6 lease request. In other words, a pool of 4 /64 network prefixes is being created: 2001:470:115:C, 2001:470:115:D, 2001:470:115:E, and 2001:470:115:F.

ipv6 dhcp server ipv6-pool1

Enable DHCPv6 on this interface, and allocate prefixes from ipv6-pool1.

Regarding the *valid lifetime* and *preferred lifetime* values above, the Cisco documentation states that the *valid lifetime* is the number of seconds the delegation remains valid before it must be renewed. The *preferred lifetime* is the number of seconds that the prefix is the “preferred prefix” for the leasing client. For testing purposes, these values were set artificially low in order to observe what happens when multiple clients keep acquiring / renewing / releasing leases. Observations show that the leasing client will issue a lease renewal request every *valid lifetime* seconds. The *preferred lifetime* only seems to come into play when a pool has been exhausted and the router must reclaim a prefix and delegate it to another client. At that point, a client which is still within the *preferred lifetime* of its lease is less likely to have its lease terminated and prefix reassigned.

Again, the lifetimes specified above are artificially (ludicrously?) low so competition for leases can be observed. More realistic numbers would something like one month for *valid lifetime* and a week for *preferred lifetime*. These numbers would essentially guarantee a customer that their prefix would not change in less than a week, and they would not need to renew their lease for a month.

In a production environment, the pools would be created large enough so every single legitimate IPv6 customer could have a valid lease claimed at the same time. Capacity planning should guarantee that no customer ever “pirates” the prefix of another customer. The purpose of the lifetimes would be to allow prefixes to be automatically reclaimed when customers cancel.

### 2.1.2 Configuring DHCPv6 Client

The KAME DHCPv6 client, dhcp6c, was chosen as the IPv6 DHCP client. That was chiefly because when this effort started, it was the only full DHCPv6 implementation available. Other DHCPv6 implementations are now available, and would probably work fine, too. During the course of this work, the KAME DHCPv6 client was ported to Linux, and some customizations were made to the source code.

Here are the steps to configure a FreeBSD or Linux box to act as a DHCPv6 client and as an IPv6 gateway device:

- 1) Configure the box to act as a gateway instead of an end node.

This is a simple matter of executing two “sysctl” commands. In FreeBSD:

```
sysctl -w net.inet6.ip6.forwarding=1
sysctl -w net.inet6.ip6.accept_rtadv=0
```

In Linux:

```
sysctl -w net.ipv6.conf.all.forwarding=1
sysctl -w net.ipv6.conf.all.accept_rtadv=0
```

This tells the operating system to act as a gateway (forward packets), and to no longer listen to route advertisements.

2) Configure and execute the DHCPv6 client.

Here is the `dhcp6c.conf` file used to configure the DHCPv6 client:

```
id-assoc pd 0 {
    prefix-interface eth1 {
        sla-id 0;
        sla-len 0;
    };
};

interface eth0 {
    send ia-pd 0;
};
```

Basically, this configuration file tells `dhcp6c` to send prefix delegation requests out interface `eth0`. It then takes the prefix thusly obtained and configures that prefix on interface `eth1`. `Dhcp6c` is executed via this command line:

```
dhcp6c -d eth0
```

3) Configure and execute the route advertising daemon.

There are two distinctly different route advertising daemon implementations. In FreeBSD it is called `rtadvd`. In the common case, `rtadvd` just “does the right thing”, and no configuration is necessary. Given an interface on the command line, it detects any prefixes which are configured on that interface and broadcasts router advertisements for that prefix out the interface. On FreeBSD, `rtadvd` was started via this command: `rtadvd xl1`

Linux uses a different route advertisement daemon, `radvd`. It is a little less sophisticated than KAME’s `rtadvd` and requires a configuration file which specifies the prefix(es) to be advertised. Once `dhcp6c` has been executed and a prefix assigned, the `ip` command may be used to determine the allocated prefix:

```
ip -6 addr show dev eth1 | grep -vi fe80:
```

That prefix is then entered into the `radvd.conf` file. For example:

```
interface eth1
{
    AdvSendAdvert on;
```

```
    prefix 2001:470:115:c026::/64 {};  
};
```

Radvd may then be executed with a command line something like this:  
`radvd -C /etc/radvd.conf`

## 2.2 Configuring DHCPv6 Over Tunnel

This was similar to configuring DHCPv6 over a LAN interface, but involved creating tunnel interfaces on both the Cisco and Linux / FreeBSD gateway device, then directing the DHCPv6 traffic and IPv6 network prefix routes over the tunnel.

### 2.2.1 Configuring the Cisco

This involved creating a new virtual interface on the Cisco, a *tunnel interface*. A new DHCP pool was also created for use with the newly created tunnel interface.

```
ipv6 dhcp pool ipv6-pool2  
prefix-delegation pool ipv6-pool2 lifetime 900 300  
exit  
ipv6 local pool ipv6-pool2 2001:470:115:10::/62 64  
  
interface Tunnel5  
no ip address  
no ip redirects  
ipv6 enable  
ipv6 dhcp server ipv6-pool2  
tunnel source 209.179.5.34  
tunnel mode ipv6ip auto-tunnel  
exit  
  
ipv6 route ::/96 Tunnel5
```

The creation of the DHCP pool was the same as for the LAN configuration. In fact, the Cisco may be configured to share the same pool for both interfaces (FastEthernet0/1 and tunnel5). The decision to create and maintain separate pools was completely arbitrary.

The most crucial step in creation of the tunnel interface was the choice of tunnel mode:

```
tunnel mode ipv6ip auto-tunnel
```

This instructs the Cisco to automatically create an IPv6 over IPv4 tunnel (IPv4 protocol 41) to the destination whenever the target IPv6 address starts with 96 “0” bits. Such an address would look like this:

```
::4.8.225.218
```

This type of IPv6 address is special. It is called an “IPv4 compatible IPv6 address”. When this type of tunnel sends data to an IPv4 compatible IPv6 address, it knows that the IPv4 endpoint of the tunnel is the last 32 bits of the IPv6 address. It can therefore automatically create a tunnel.

The statement `ipv6 route ::/96 Tunnel5` then instructs the router to route all traffic to such addresses out the tunnel5 interface.

### 2.2.2 Configuring the FreeBSD Tunnel

Here are the commands used to configure the tunnel on FreeBSD and obtain the IPv6 prefix via DHCPv6:

```
route delete -inet6 ::/96
gifconfig gif0 4.8.225.218 209.179.5.34
ifconfig gif0 inet6 ::4.8.225.218 prefixlen 96 up
ifconfig gif0 inet6 delete fe80::240:63ff:fed5:982e
dhcp6c -d gif0
route add -inet6 default ::209.179.5.34
```

A detailed explanation:

```
route delete -inet6 ::/96
```

For some reason, when FreeBSD boots up all traffic destined to IPv4 compatible IPv6 addresses are routed to the loopback interface. This statement deletes that route. Otherwise, no traffic will ever go out the tunnel interface.

```
gifconfig gif0 4.8.225.218 209.179.5.34
```

This creates the tunnel interface. 4.8.225.218 is the IPv4 address of the FreeBSD box. 209.179.5.34 is the IPv4 address of the Cisco.

```
ifconfig gif0 inet6 ::4.8.225.218 prefixlen 96 up
```

This configures the IPv4 address on the gif0 interface. The address is the IPv4 compatible IPv6 version of its IPv4 address. This causes all packets going out this interface to have a source address of ::4.8.225.218. The Cisco will then know how to route back to this interface using the automatic tunnel interface tunnel5 described above.

```
ifconfig gif0 inet6 delete fe80::240:63ff:fed5:982e
```

This is not intuitive. When an IPv6 interface is initially brought up on FreeBSD, a *link-local* address is automatically configured. The problem with that is that it will be used as the source address for all link-local operations. Since DHCPv6 uses link-local broadcast messages to solicit the router for its configuration, it will end up using this link-local address as its source address. The Cisco would then be unable to route messages back, as it would be attempting to send the DHCPv6 messages out a local interface to a link-local address. By deleting this automatically configured link-local address, the only remaining address on the gif0 interface is ::4.8.225.218, which the Cisco will be able to route back to via the tunnel5 interface.

```
dhcp6c -d gif0
```

This starts the DHCPv6 client daemon, directing it to use gif0 for its traffic.

```
route add -inet6 default ::209.179.5.34
```

This adds a default route out the gif0 interface. All subsequent IPv6 traffic will now route out this interface.

### 2.2.3 Configuring the Linux Tunnel

Here are the commands used to configure the tunnel on Linux and obtain the IPv6 prefix via DHCPv6. This is substantially similar to the steps taken on FreeBSD, but there are some differences which will be discussed below.

```
ip tunnel add gif0 mode sit ttl 200 remote 209.179.5.34 local 4.8.225.218
ip link set dev gif0 mtu 1440 up
ip -6 addr add ::4.8.225.218/96 dev gif0
dhcp6c -d -s ::4.8.225.218 gif0
ip -6 route add default dev gif0
```

A detailed explanation:

It should be observed that the command `route delete -inet6 ::/96` (or its Linux equivalent) is not present. It is not needed as no such route gets configured into the Linux kernel by default.

Also, there is no Linux equivalent to the command

```
ifconfig gif0 inet6 delete fe80::240:63ff:fed5:982e
```

Depending upon the exact version of the Linux kernel, such an operation was not permitted. To work around that, the source to `dhcp6c` was modified to implement a new command line option, “-s”, which permits the source address of all outbound packets to be specified.

```
ip tunnel add gif0 mode sit ttl 200 remote 209.179.5.34 local 4.8.225.218
```

This creates the tunnel interface. 4.8.225.218 is the IPv4 address of the Linux box. 209.179.5.34 is the IPv4 address of the Cisco.

```
ip link set dev gif0 mtu 1440 up
```

Bring the gif0 interface up and specify the MTU. By default, Linux chooses an MTU of 1448, which does not allow for the header overhead of IPv6-in-IP.

```
ip -6 addr add ::4.8.225.218/96 dev gif0
```

This configures the IPv4 address on the gif0 interface. The address is the IPv4 compatible IPv6 version of its IPv4 address.

```
dhcp6c -d -s ::4.8.225.218 gif0
```

This starts the DHCPv6 client daemon, directing it to use gif0 for its traffic. As noted above, the source to `dhcp6c` has been modified so the source address of all outbound packets may be specified.

```
ip -6 route add default dev gif0
```

This adds a default route out the gif0 interface. All subsequent IPv6 traffic will now route out this interface.

### 3 Authentication using IPSEC

The method described above of using IPv6-over-IPv4 tunnels and DHCPv6 works well to automatically allocate IPv6 prefixes to end users. It has one major drawback, however: it is not authenticated. Any user who knows the IP address of the DHCPv6 server may make a DHCP request and get a prefix allocated to them.

The decision was made to try and use IPSEC for authentication. Specifically, IPSEC's "Authentication Header" (AH) was chosen. AH does not encrypt traffic, it merely authenticates it by applying a hash algorithm to packet headers. This means that it is very lightweight and does not place a heavy load on routers.

Cisco's implementation of IPSEC permits a RADIUS system to be used for backend key storage and retrieval. This was also considered a major positive as it leveraged a technology already used by EarthLink for authenticating end-users.

The system was configured to work in this manner: A tunnel interface was configured on the Cisco. DHCPv6 was configured to operate on the tunnel interface. The tunnel interface was further configured to reject any packets which were not protected by IPSEC Header Authentication. So no DHCPv6 requests succeed until the peer (the end-user's CPE) first established an IPSEC "Security Association" with the Cisco.

Upon initiation of an IPSEC session by the CPE, the end-user's email address would be supplied as the unique identifier for the association. The Cisco could then use that email address to make a RADIUS request to a backend RADIUS server and extract the shared key. That key is used to establish the security association.

#### 3.1 Configuring the Cisco for IPSEC

To say that configuring IPSEC on a Cisco is non-intuitive is an enormous understatement. The following steps were required:

1. Configure RADIUS authentication and enable it for IPSEC.
2. Configure IKE / ISAKMP (Internet Key Exchange / Internet Security Association and Key Management Protocol)
3. Create a *Transform Set* specifying Authenticated Header (AH) mode and the hash to be used.
4. Create a *Dynamic Crypto Map* and apply the transform set to it.
5. Create an access list specifying which traffic is to be protected by IPSEC
6. Create a *Static Crypto Map* and tell it to use the dynamic crypto map.
7. Apply the static map to an interface.

Here are the actual commands used to configure the Cisco:

```
aaa new-model
aaa authorization network list_ipsec1 group radius
radius-server host 209.179.5.35 auth-port 1812 acct-port 1813
radius-server host 209.179.5.35 key 0 my-secret
!
crypto isakmp policy 10
  encr 3des
  authentication pre-share
  group 2
  lifetime 5400
!
crypto ipsec transform-set tf_ipsec1 ah-md5-hmac
mode transport require
!
crypto dynamic-map dm_ipsec1 10
  set security-association lifetime seconds 14400
  set transform-set tf_ipsec1
  set pfs group2
  match address 101
!
crypto map sm_ipsec1 isakmp authorization list list_ipsec1
crypto map sm_ipsec1 100 ipsec-isakmp dynamic dm_ipsec1
!
interface FastEthernet0/0
  crypto map sm_ipsec1
```

A detailed explanation:

```
aaa new-model
aaa authorization network list_ipsec1 group radius
radius-server host 209.179.5.35 auth-port 1812 acct-port 1813
radius-server host 209.179.5.35 key 0 my-secret
```

These commands enable and configure RADIUS authentication. An authorization list named “list\_ipsec1” is created, which will be used in later configuration directives.

```
crypto isakmp policy 10
  encr 3des
  authentication pre-share
  group 2
  lifetime 5400
```

These statements configure IKE / ISAKMP. 3DES will be used as the encryption algorithm. Pre-shared keys will be used to establish the IKE security association. Diffie-Hellman Group 2 (1024 bits) is used to negotiate the exchange. The keys will be re-negotiated every 5400 seconds (90 minutes).

```
crypto ipsec transform-set tf_ipsec1 ah-md5-hmac
mode transport require
```

These statements create a *Transform Set* named tf\_ipsec1. AH (Authenticated Header) mode will be employed as opposed to ESP (Encapsulating Security Payload). This means that packet contents will not be encrypted, merely authenticated. Transport mode will be used instead of tunnel mode, chiefly because it places less load on the router.

```
crypto dynamic-map dm_ipsec1 10
set security-association lifetime seconds 14400
set transform-set tf_ipsec1
set pfs group2
match address 101
```

Create a dynamic crypto map named dm\_ipsec1. The lifetime is set to 4 hours, meaning new keys will be negotiated every 4 hours. The transform set tf\_ipsec1 defined above will be used. PFS (Perfect Forward Secrecy) will be used, meaning that when new keys are negotiated, the old keying material will not be used so that compromising one key will not cause future keys to be compromised. Diffie-Hellman group 2 (1024 bits) will be used for PFS. Access list number 101 will be used to define which traffic is to be protected by IPSEC.

```
crypto map sm_ipsec1 isakmp authorization list list_ipsec1
crypto map sm_ipsec1 100 ipsec-isakmp dynamic dm_ipsec1
```

These two commands create a static map named sm\_ipsec1. This is necessary because only static crypto maps may be applied to interfaces. The first line causes RADIUS to be used to get the initial key during IKE / ISAKMP association establishment. The second line causes the static map to refer to the dynamic map created above.

```
interface FastEthernet0/0
crypto map sm_ipsec1
```

These lines apply this entire mess to interface FastEthernet0/0. That interface is the “world-facing” interface. All traffic coming in from the outside world passes through that interface.

The final topic which has not been discussed is the ACL which was created to define which traffic would be IPSEC protected. That ACL is:

```
access-list 101 deny ospf host 209.179.5.33 host 209.179.5.34
access-list 101 deny ospf host 209.179.5.34 host 209.179.5.33
access-list 101 deny ip host 209.179.5.35 host 209.179.5.34
access-list 101 deny ip host 209.179.5.34 host 209.179.5.35
access-list 101 permit ip any host 209.179.5.34
access-list 101 permit ip host 209.179.5.34 any
```

In this usage, “deny” means that IPSEC will not be applied to the traffic and “permit” means that IPSEC will be applied. 209.179.5.34 is the address of the R&D router. 209.179.5.33 is the address of the default router through which all internet-facing traffic flows. OSPF between those two routers is permitted without requiring IPSEC protection. 209.179.5.35 is the address of the RADIUS server, and to make implementation and debugging easier IPSEC protection was not applied to traffic between the Cisco and the RADIUS server. Other than those few exceptions, all traffic to and from 209.179.5.34 is to be protected by IPSEC. No outside machine can communicate with the router unless it has first established an IPSEC security association.

## 4 IPSEC Peer Configuration

The steps required to configure the other side of the IPSEC association vary widely depending upon the operating system and IPSEC implementation being used. To date, four different implementations have been successfully tested:

1. FreeBSD using the KAME IPv6 / IPSEC stack
2. Linux kernel 2.6 using USAGI IPv6 stack
3. Linux kernel 2.4 using USAGI and FreeS/WAN IPSEC implementation
4. A Linksys WRT54G Wireless Access Point with a specially crafted firmware image using custom patches derived from USAGI and FreeS/WAN

FreeBSD and USAGI Linux 2.6 are very nearly identical. The Linux 2.4 implementation of FreeS/WAN and the implementation on the WRT54G should have been identical, but due to flash memory constraints the entire FreeS/WAN implementation could not be installed on the WRT54G. So a stripped down version of FreeS/WAN was installed and the low-level IPSEC commands were used instead of the more friendly shell scripts normally used in a FreeS/WAN environment.

Here are the implementation / configuration details:

### 4.1 FreeBSD / USAGI Linux 2.6

There are two fundamental tasks required: configuring racoon, the IKE / ISAKMP daemon; and manipulating the Security Association Database (SAD) and Security Policy Database (SPD) using the “setkey” command.

#### 4.1.1 Configuring racoon

Here is the racoon.conf file that was used:

```
remote anonymous
{
    exchange_mode aggressive;
    my_identifier user_fqdn "rndtest1@research.earthlink.net";
```

```

lifetime time 24 hour ;
proposal {
    encryption_algorithm 3des;
    hash_algorithm sha1;
    authentication_method pre_shared_key ;
    dh_group 2 ;
}
proposal_check obey;
}
sainfo anonymous
{
    pfs_group 2;
    lifetime time 4 hour ;
    encryption_algorithm 3des ;
    authentication_algorithm hmac_md5 ;
    compression_algorithm deflate ;
}

```

The “remote anonymous” section defines parameters used during IKE key negotiations. A couple of the items bear comment:

- “Aggressive Mode” is used during key exchange. This is necessary so the user\_fqdn is sent in the first packet of the key exchange. That, in turn, is necessary so the router knows the email address of the user so the proper key can be extracted from RADIUS.
- “proposal\_check obey” means that this IPSEC peer will obey whatever is proposed by the other peer. Mainly this is related to the “lifetime” parameter. A 24 hour lifetime is specified in the proposal (the maximum permitted by RFC). The router is configured for a much shorter lifetime (90 minutes). By specifying “proposal\_check obey” we are agreeing to accept whatever lifetime is proposed by the router.

The “sainfo anonymous” section defines the attributes which will be used to establish the IPSEC Security Association. The attributes listed here match up with those defined in 4.1 above, albeit in a less convoluted and less confusing manner than that devised by Cisco.

#### 4.1.2 Manipulating Security Policy Database

FreeBSD maintains a kernel table called the *Security Policy Database* (SPD) which instructs the kernel as to which packets should be protected by IPSEC and how the packets should be manipulated. The kernel SPD is manipulated via the “setkey” command.

The goal is to have all packets which are IPv6-in-IPv4 packets going to or from IP address 209.179.5.34 be IPSEC protected. So the following lines were put into a file named “policy.template”:

```

spdadd %MYADDR%/32 209.179.5.34/32 41 -P out ipsec ah/transport//require;
spdadd 209.179.5.34/32 %MYADDR%/32 41 -P in ipsec ah/transport//require;

```

These two lines tell the kernel to use IPSEC on all packets going to or from 209.179.5.34, and which are protocol 41 (IPv6 over IPv4.) Packets matching that criteria are to be protected using Authenticated Header (AH) and transport mode.

A simple shell script was then used to replace “%MYADDR%” with the actual IP address of the machine and pass the result to the setkey command:

```
MYIP=$(ifconfig x10 | awk '/inet /{print $2}')
```

```
sed "s/%MYADDR%/$MYIP/g" policy.template | setkey -c
```

There is no command in FreeBSD to explicitly force an IPSEC security association to be established. FreeBSD will automatically attempt to establish a security association the first time traffic matching the SPD is encountered.

## 4.2 Linux 2.4 / FreeS/WAN

The Linux 2.4 kernel does not have any IPSEC support. To provide IPSEC functionality, the FreeS/WAN system was used, specifically super-freeswan-1.99.8. Super-freeswan was needed because it contains a patch to the original FreeS/WAN code implementing *Aggressive Mode* during IKE key negotiation. Aggressive mode is required in order to permit the Cisco router to obtain the shared key via RADIUS.

The FreeS/WAN system is extremely powerful and is capable of supporting some very complex configurations. For our purposes, however, configuration turned out to be quite simple. Two configuration files are required: ipsec.conf and ipsec.secrets. Ipsec.conf contains configuration directives, and ipsec.secrets contains shared secrets and RSA keys. In this specific case, only shared secrets are stored in the ipsec.secrets file.

The ipsec.conf file looks like this:

```
config setup
    interfaces="%defaultroute"
    klipsdebug=none
    plutodebug=none

conn cisco
    left=%defaultroute
    leftid=rndtest1@research.earthlink.net
    right=209.179.5.34
    rightid=209.179.5.34
    authby=secret
    agrmode=yes
    pfs=yes
    type=transport
    auth=ah
    keyingtries=5
```

The above configuration file defines a single IPSEC connection named “cisco”. *Aggressive mode* is enabled, as is *Perfect Forward Secrecy*. IPSEC *Transport Mode* will be used. The connection will be protected via *Authenticated Header* mode.

FreeS/WAN will try at most 5 times to establish the security association before it gives up.

The ipsec.secrets file contains only a single line:

```
rndtest1@research.earthlink.net 209.179.5.34: PSK "my-secret"
```

The first column must match the value of “leftid” in the ipsec.conf file. The 2nd column must match the value of “rightid”. “PSK” means that Private Shared Keys will be used. The final column is the shared secret, surrounded by quotes.

Unlike FreeBSD, FreeS/WAN does not establish an IPSEC security association automatically. The following three commands must be executed to establish the security association:

```
ipsec setup --start      # Initialize the IPSEC system
ipsec auto --add cisco  # Load info for connection named cisco
ipsec auto --up cisco   # Initiate IPSEC security association
```

### 4.3 Linux 2.4 on the Linksys WRT54G

As mentioned above, running the FreeS/WAN system on the WRT54G should, in theory, have been identical to running FreeS/WAN on any other Linux system based on the 2.4 kernel. That proved not to be the case.

The FreeS/WAN system relies heavily on a complex web of sophisticated shell scripts. When the “ipsec setup” and “ipsec auto” commands listed above are executed, a lot of “magic” happens and the user is shielded from a great amount of complexity. Unfortunately, the shell provided on the WRT54G is a very stripped down shell, and it does not support a lot of shell-language features required by the FreeS/WAN shell scripts. Additionally, due to constraints on available flash memory on the WRT54G, it was not possible to install the entire FreeS/WAN system. It was necessary to debug the FreeS/WAN system to find out exactly what commands were being issued by the shell script wrappers, then create a much simpler script which issued those commands. Here are those commands:

```
echo 0 > /proc/sys/net/ipv4/conf/$IFACE/rp_filter
tncfg --attach --virtual ipsec0 --physical $IFACE
ifconfig ipsec0 inet $MYIP netmask $MASK
pluto --adns /sbin/_pluto_adns
whack --name cisco --pfs --authenticate --ike "3des" \
      --aggrmode --disablearrivalcheck --psk \
      --ikelifetime 5400 --rekeywindow 540 --rekeyfuzz 25 \
      --keyingtries 5 --host "$MYIP" --nexthop "$MYIP" \
      --id "$IPSEC_ID" --to --host "$IPSEC_PEER" \
      --nexthop "$DEFAULT" --id "$IPSEC_PEER" \
      --ipseclifetime 14400
whack --listen
whack --name cisco --initiate
route add -host "$IPSEC_PEER" gw $MYIP dev ipsec0
```

The shell variables listed above have the following values / meanings:

**IFACE** The interface associated with the default route. On the WRT54G this is typically vlan1 (when using DHCP) or ppp0 (when using PPPOE).

**MYIP** The IPv4 address assigned to the WRT54G.

**MASK** The netmask.

**IPSEC\_ID** The FQDN (rndtest1@research.earthlink.net in these examples.)

**IPSEC\_PEER** The IP address of the Cisco router (209.179.5.34).

**DEFAULT** The default route.

## 5 RADIUS Server Details

Details on how to configure and run a RADIUS server are beyond the scope of this document. There are a few details, however, that do bear mention.

When the Cisco authenticates against the RADIUS server, it supplies the IPSEC FQDN as the username, and always uses the hardcoded password “cisco”. It expects the following attributes to be supplied in RADIUS reply packet:

- Service-Type
- Tunnel-Type
- Tunnel-Medium-Type
- Cisco-AVPair

Here is a sample entry from the *users* file of a RADIUS server running FreeRADIUS version 1.0.2:

```
rndtest1@research.earthlink.net Auth-Type := Local, Password == cisco
    Service-Type = Outbound-User,
    Tunnel-Type = :1:AH,
    Tunnel-Medium-Type = :1:IP,
    Cisco-AVPair="ipsec:tunnel-password=MY-IPSEC-SECRET",
    Cisco-AVPair+="ipsec:key-exchange=ike"
```

The text “MY-IPSEC-SECRET” would be replaced with the actual shared IPSEC secret.

A sample entry for the *clients.conf* file is shown below. Specifying “cisco” as the *nastype* is required. That is how FreeRADIUS knows how to interpret the attribute named *Cisco-AVPair*.

```
client 209.179.5.34 {
    secret          = some-shared-secret
    shortname      = my-client-name
    nastype        = cisco
}
```