IPsec and IKE Administration Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817–2694–10
December 2003

030930@6671

Contents

Preface

The *IPsec and IKE Administration Guide* updates Chapters 19, 20, and 21 of the *System Administration Guide: IP Services*. This book assumes the following:

- You have already installed the SunOS™ 5.9 operating environment.
- You have updated the SunOS 5.9 operating environment with the Solaris 9 12/03 release.
- You have set up any networking software that you plan to use.

The SunOS 5.9 operating environment is part of the Solaris™ product family, which also includes the Solaris Common Desktop Environment (CDE). The SunOS 5.9 operating environment is compliant with AT&T's System V, Release 4 operating system.

Note – The Solaris operating environment runs on two types of hardware, or platforms—SPARC® and x86. The Solaris operating environment runs on both 64-bit address spaces and 32-bit address spaces. The information in this document pertains to both platforms and both address spaces. Exceptions are called out in a special chapter, section, note, bullet, figure, table, example, or code example.

Who Should Use This Book

This book is intended for anyone responsible for administering one or more systems that run the Solaris 9 release. To use this book, you should have one year or two years of UNIX® system administration experience. A UNIX system administration training course might be helpful.

How This Book Is Organized

Chapter 1 provides an overview of IP Security Architecture (IPsec). IPsec provides protection for IP datagrams.

Chapter 2 provides procedures for implementing IPsec on your network.

Chapter 3 provides an overview of Internet Key Exchange (IKE) for use with IPsec.

Chapter 4 provides procedures for implementing IKE.

Appendix A provides a list of changes between the Solaris 9 release and the Solaris 9 12/03 release.

The Glossary provides definitions of key IP security terms.

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

TABLE P–1 Typographic Conventions *(Continued)*

Typeface or Symbol	Meaning	Example
AaBbCc123	What you type, contrasted with onscreen computer output	`machine_name% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **rm** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

IP Security Architecture (Overview)

The IP security architecture (IPsec) provides cryptographic protection for IP datagrams in IPv4 and IPv6 network packets. This protection can include confidentiality, strong integrity of the data, data authentication, and partial sequence integrity. Partial sequence integrity is also known as replay protection.

IPsec is performed inside the IP module. IPsec can be applied with or without the knowledge of an Internet application. When used properly, IPsec is an effective tool in securing network traffic.

This chapter contains the following information:

For instructions on implementing IPsec on your network, see Chapter 2.

Introduction to IPsec

IPsec provides security mechanisms that include secure datagram authentication and encryption mechanisms within IP. When you invoke IPsec, IPsec applies the security mechanisms to IP datagrams that you have enabled in the IPsec global policy file. Applications can invoke IPsec to apply security mechanisms to IP datagrams on a per-socket level.

Figure 1–1 shows how an IP addressed packet, as part of an IP datagram, proceeds when IPsec has been invoked on an outbound packet. As you can see from the flow diagram, authentication header (AH) and encapsulating security payload (ESP) entities can be applied to the packet. Subsequent sections describe how you apply these entities, as well as authentication and encryption algorithms.

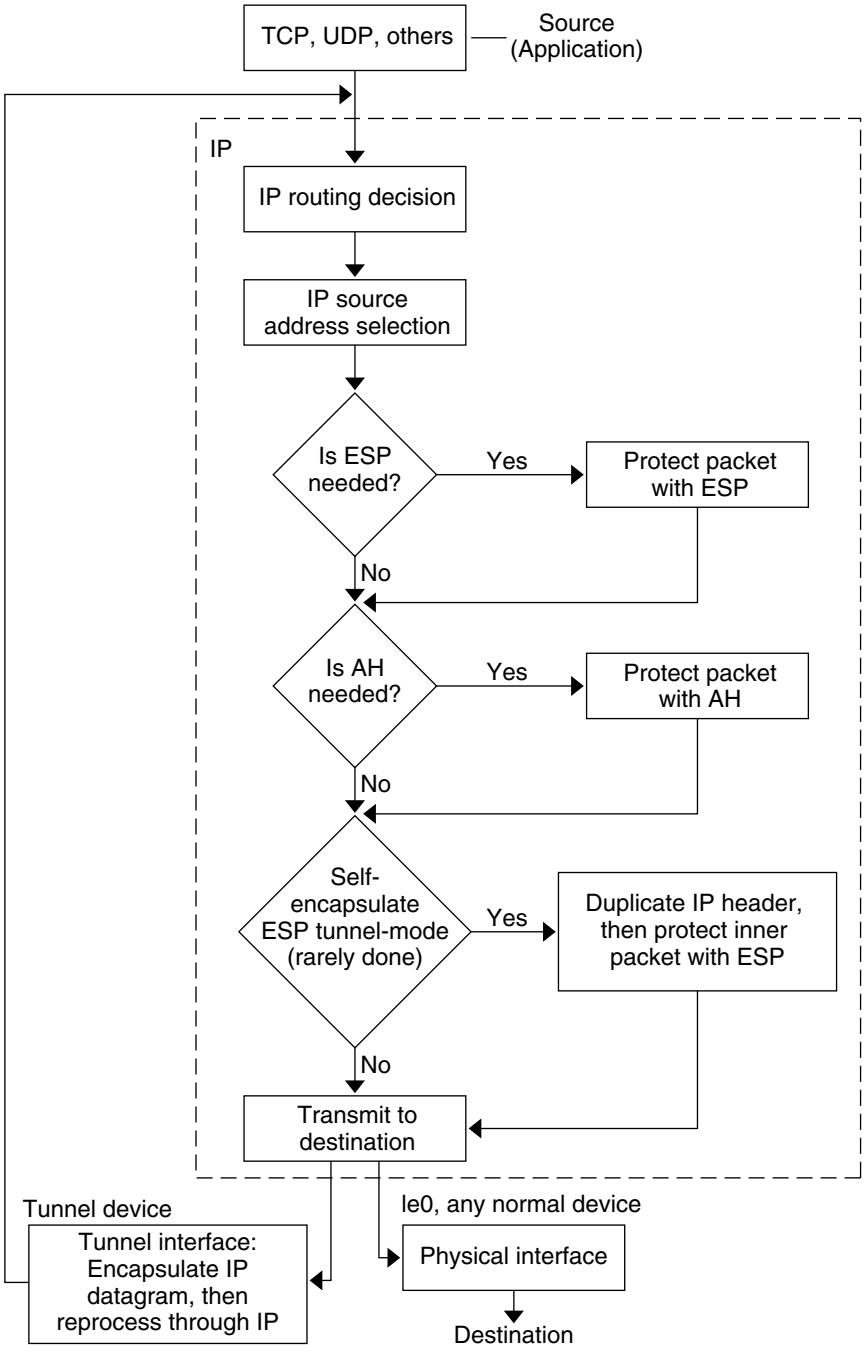


FIGURE 1–1 IPsec Applied to Outbound Packet Process

Figure 1–2 shows the IPsec inbound process.

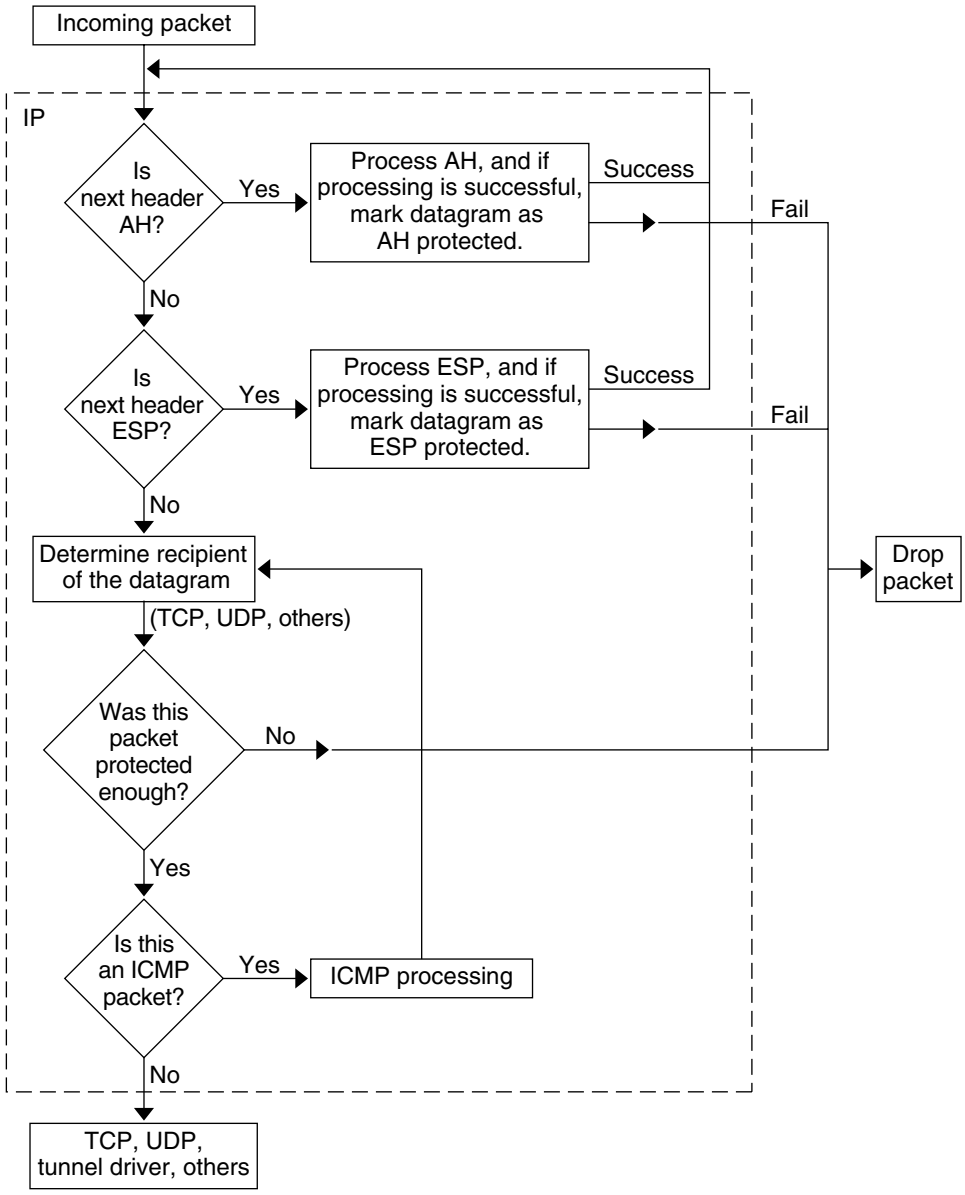


FIGURE 1–2 IPsec Applied to Inbound Packet Process

IPsec Security Associations

An IPsec security association (SA) specifies security properties that are recognized by communicating hosts. These hosts typically require two SAs to communicate securely. A single SA protects data in one direction. The protection is either to a single host or a group (multicast) address. Because most communication is peer-to-peer or client-to-server, two SAs must be present to secure traffic in both directions.

The security protocol (AH or ESP), destination IP address, and security parameter index (SPI) identify an IPsec SA. The SPI, an arbitrary 32-bit value, is transmitted with an AH or ESP packet. The `ipsecah`(7P) and `ipsecesp`(7P) man pages explain the extent of protection that is provided by AH and ESP. An integrity checksum value is used to authenticate a packet. If the authentication fails, the packet is dropped.

Security associations are stored in a security associations database. A socket-based administration engine, the `pf_key` interface, enables privileged applications to manage the database. The `in.iked` daemon provides automatic key management. See the `pf_key`(7P) and `in.iked`(1M) man pages.

Key Management

A security association contains the following information:

- Material for keys for encryption and authentication
- The algorithms that can be used
- The identities of the endpoints
- Other parameters that are used by the system

SAs require keying material for authentication and encryption. The managing of keying material that SAs require is called key management. The Internet Key Exchange (IKE) protocol handles key management automatically. You can also manage keys manually with the `ipseckey` command. SAs on IPv4 and IPv6 packets can use automatic key management.

See “IKE Overview” on page 50, for how IKE manages cryptographic keys automatically. See “Keying Utilities” on page 25, for how you can manually manage the cryptographic keys by using the `ipseckey` command. The `ipseckey`(1M) man page provides a detailed description of the command options.

Protection Mechanisms

IPsec provides two mechanisms for protecting data:

- Authentication Header (AH)
- Encapsulating Security Payload (ESP)

Both mechanisms have their own Security Association Database (SADB).

Authentication Header

The authentication header provides data authentication, strong integrity, and replay protection to IP datagrams. AH protects the greater part of the IP datagram. AH cannot protect fields that change nondeterministically between sender and receiver. For example, the IP TTL field is not a predictable field and, consequently, not protected by AH. AH is inserted between the IP header and the transport header. The transport header can be TCP, UDP, ICMP, or another IP header when tunnels are being used. See the `tun`(7M) man page for details on tunneling.

Authentication Algorithms and the AH Module

IPsec implements AH as a module that is automatically pushed on top of IP. The `/dev/ipsecah` entry tunes AH with the `ndd` command. Future authentication algorithms can be loaded on top of AH. Current authentication algorithms include HMAC-MD5 and HMAC-SHA-1. Each authentication algorithm has its own key size and key format properties. See the `authmd5h`(7M) and `authsha1`(7M) man pages for details. For tuning IP configuration parameters, see the `ndd`(1M) man page.

Security Considerations for AH

Replay attacks threaten an AH when an AH does not enable replay protection. An AH does not protect against eavesdropping. Adversaries can still see data that is protected with AH.

Encapsulating Security Payload

The encapsulating security payload (ESP) header provides confidentiality over what the ESP encapsulates, as well as the services that AH provides. However, ESP only provides its protections over the part of the datagram that ESP encapsulates. ESP's authentication services are optional. These services enable you to use ESP and AH together on the same datagram without redundancy. Because ESP uses encryption-enabling technology, ESP must conform to U.S. export control laws.

ESP encapsulates its data, so ESP only protects the data that follows its beginning in the datagram. In a TCP packet, ESP encapsulates only the TCP header and its data. If the packet is an IP-in-IP datagram, ESP protects the inner IP datagram. Per-socket policy allows *self-encapsulation*, so ESP can encapsulate IP options when ESP needs to. Unlike the authentication header (AH), ESP allows multiple kinds of datagram protection. Using only a single form of datagram protection can make the datagram vulnerable. For example, if you use ESP to provide confidentiality only, the datagram is still vulnerable to replay attacks and cut-and-paste attacks. Similarly, if ESP protects only integrity, ESP could provide weaker protection than AH. The datagram would be vulnerable to eavesdropping.

Algorithms and the ESP Module

IPsec ESP implements ESP as a module that is automatically pushed on top of IP. The `/dev/ipsecesp` entry tunes ESP with the `ndd` command. ESP allows encryption algorithms to be pushed on top of ESP, in addition to the authentication algorithms that are used in AH. Encryption algorithms include Data Encryption Standard (DES), Triple-DES (3DES), Blowfish, and AES. Each encryption algorithm has its own key size and key format properties. Because of export laws in the United States and import laws in other countries, not all encryption algorithms are available outside of the United States. For tuning IP configuration parameters, see the `ndd`(1M) man page.

Security Considerations for ESP

An ESP without authentication is vulnerable to cut-and-paste cryptographic attacks and to replay attacks. When you use ESP without confidentiality, ESP is as vulnerable to eavesdropping as AH is.

Authentication and Encryption Algorithms

IPsec uses two types of algorithms, authentication and encryption. The authentication algorithms and the DES encryption algorithms are part of core Solaris installation. If you plan to use other algorithms that are supported for IPsec, you must install the Solaris Encryption Kit. The Solaris Encryption Kit is provided on a separate CD.

Authentication Algorithms

Authentication algorithms produce an integrity checksum value or *digest* that is based on the data and a key. The man pages for authentication algorithms describe the size of both the digest and key. The following table lists the authentication algorithms that are supported in the Solaris operating environment. The table also lists the format of the algorithms when the algorithms are used as security options to the IPsec utilities and their man page names.

TABLE 1–1 Supported Authentication Algorithms

Algorithm Name	Security Option Format	Man Page
HMAC-MD5	md5, hmac-md5	`authmd5h`(7M)
HMAC-SHA-1	sha, sha1, hmac-sha, hmac-sha1	`authsha1`(7M)

Encryption Algorithms

Encryption algorithms encrypt data with a key. The algorithms operate on data in units of a *block size*. The man pages for encryption algorithms describe the block size and the key size for each algorithm. By default, the DES–CBC and 3DES-CBC algorithms are installed.

The AES and Blowfish algorithms are available to IPsec when you install the Solaris Encryption Kit. The kit is available on a separate CD that is *not* part of the Solaris 9 installation box. The *Solaris 9 Encryption Kit Installation Guide* describes how to install the Solaris Encryption Kit.

The following table lists the encryption algorithms that are supported in the Solaris operating environment. The table lists the format of the algorithms when the algorithms are used as security options to the IPsec utilities. The table also lists their man page names, and lists the package that contains the algorithm.

TABLE 1–2 Supported Encryption Algorithms

Algorithm Name	Security Option Format	Man Page	Package
DES-CBC	des, des-cbc	`encrdes`(7M)	SUNWcsr, SUNWcarx.u
3DES–CBC or Triple-DES	3des, 3des-cbc	`encr3des`(7M)	SUNWcsr, SUNWcarx.u
Blowfish	blowfish, blowfish-cbc	`encrbfsh`(7M)	SUNWcryr, SUNWcryrx
AES-CBC	aes, aes-cbc	`encraes`(7M)	SUNWcryr, SUNWcryrx

Protection Policy and Enforcement Mechanisms

IPsec separates its protection policy from its enforcement mechanisms. You can enforce IPsec policies in the following places:

- On a system-wide level
- On a per-socket level

You use the `ipsecconf` command to configure the system-wide policy. See the `ipsecconf`(1M) man page.

IPsec applies the system-wide policy to incoming datagrams and outgoing datagrams. You can apply some additional rules to outgoing datagrams, because of the additional data that is known by the system. Inbound datagrams can be either accepted or dropped. The decision to drop or accept an inbound datagram is based on several criteria, which sometimes overlap or conflict. Conflicts are resolved by determining which rule is parsed first. Except when a policy entry states that traffic should bypass all other policy, the traffic is automatically accepted. Outbound datagrams are either sent with protection or without protection. If protection is applied, the algorithms are either specific or non-specific.

The policy that normally protects a datagram can be bypassed. You can either specify an exception in the system-wide policy, or you can request a bypass in the per-socket policy. For intra-system traffic, policies are enforced, but actual security mechanisms are not applied. Instead, the outbound policy on an intra-system packet translates into an inbound packet that has had those mechanisms applied.

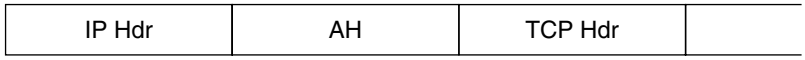
Transport and Tunnel Modes

When you invoke ESP or AH after the IP header to protect a datagram, you are using transport mode. An example follows. A packet starts off with the following header:

ESP, in transport mode, protects the data as follows:

Encrypted

AH, in transport mode, protects the data as follows:



AH actually covers the data before the data appears in the datagram. Consequently, the protection that is provided by AH, even in transport mode, covers some of the IP header.

When an entire datagram is *inside* the protection of an IPsec header, IPsec is protecting the datagram in tunnel mode. Because AH covers most of its preceding IP header, tunnel mode is usually performed only on ESP. The previous example datagram would be protected in tunnel mode as follows:

Encrypted

In tunnel mode, the inner header is protected, while the outer IP header is unprotected. Often, the outer IP header has different source and different destination addresses from the inner IP header. The inner and outer IP headers can match if, for example, an IPsec-aware network program uses self-encapsulation with ESP. Self-encapsulation with ESP protects an IP header option.

The Solaris implementation of IPsec is primarily an implementation of IPsec in transport mode. Tunnel mode is implemented as a special instance of the transport mode. The implementation treats IP-in-IP tunnels as a special transport provider. The `ifconfig` configuration options to set tunnels are nearly identical to the options that are available to socket programmers when enabling per-socket IPsec. Also, tunnel mode can be enabled in per-socket IPsec. In per-socket tunnel mode, the inner packet IP header has the same addresses as the outer IP header. For details on per-socket policy, see the `ipsec`(7P) man page. For configuring tunnels, see the `ifconfig`(1M) man page.

Trusted Tunnels

A configured tunnel is a point-to-point interface. The tunnel enables an IP packet to be encapsulated within an IP packet. A correctly configured tunnel requires both a tunnel source and a tunnel destination. For more information, see the `tun`(7M) man page and "Solaris Tunneling Interfaces for IPv6" in *System Administration Guide: IP Services*.

A tunnel creates an apparent physical interface to IP. The physical link's integrity depends on the underlying security protocols. If you set up the security associations securely, then you can trust the tunnel. Packets that exit the tunnel must have originated from the peer that was specified in the tunnel destination. If this trust exists, you can use per-interface IP forwarding to create a virtual private network.

Virtual Private Networks

You can use IPsec to construct a virtual private network (VPN). You use IPsec by constructing an Intranet that uses the Internet infrastructure. For example, an organization that uses VPN technology to connect offices with separate networks, can deploy IPsec to secure traffic between the two offices.

The following figure illustrates how two offices use the Internet to form their VPN with IPsec deployed on their network systems.

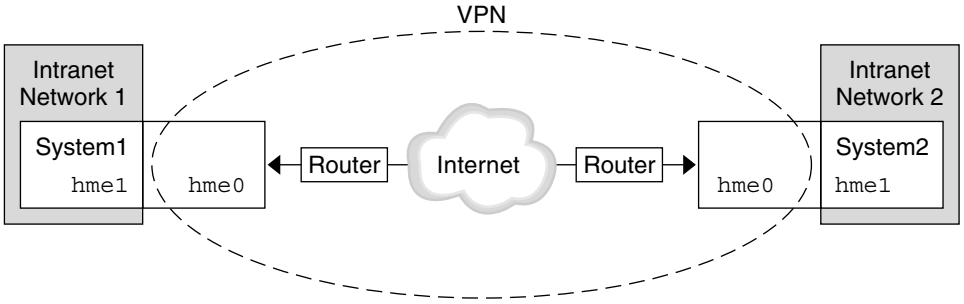


FIGURE 1–3 Virtual Private Network

See "How to Set Up a Virtual Private Network (VPN)" on page 35 for a description of the setup procedure.

IPsec Utilities and Files

This section describes the configuration file that initializes IPsec. This section also describes various commands that enable you to manage IPsec within your network. For instructions about how to implement IPsec within your network, see "Implementing IPsec (Task Map)" on page 29.

TABLE 1–3 List of Selected IPsec Files and Commands

IPsec File or Command	Description
`/etc/inet/ipsecinit.conf` file	IPsec policy file. If this file exists, IPsec is activated at boot time.
`ipsecconf` command	IPsec policy command. The boot scripts use `ipsecconf` to read the `/etc/inet/ipsecinit.conf` file and activate IPsec. Useful for viewing and modifying the current IPsec policy, and for testing.
`PF_KEY` socket interface	Interface for security association database. Handles manual and automatic key management.
`ipseckey` command	IPsec SA maintenance and keying command. `ipseckey` is a command-line front end to the `PF_KEY` interface. `ipseckey` can create, destroy, or modify security associations.
`/etc/inet/secret/ipseckeys` file	Keys for IPsec security associations. If the `ipsecinit.conf` exists, the `ipseckeys` file is automatically read at boot time.
`/etc/inet/ike/config` file	IKE configuration and policy file. If this file exists, the IKE daemon, `in.iked`, provides automatic key management. The management is based on rules and global parameters in the `/etc/inet/ike/config` file. See "IKE Utilities and Files" on page 53.

IPsec Policy Command

You use the `ipsecconf` command to configure the IPsec policy for a host. When you run the command to configure the policy, the system creates a temporary file that is named `ipsecpolicy.conf`. This file holds the IPsec policy entries that were set in the kernel by the `ipsecconf` command. The system uses the in-kernel IPsec policy entries to check all outbound and inbound IP datagrams for policy. Forwarded datagrams are not subjected to policy checks that are added by using this command. For information on how to protect forwarded packets, see the `ifconfig`(1M) and `tun`(7M) man pages. For IPsec policy options, see the `ipsecconf`(1M) man page.

You must become superuser or assume an equivalent role to invoke the `ipsecconf` command. The command accepts entries that protect traffic in both directions, and entries that protect traffic in only one direction.

Policy entries with a format of local address and remote address can protect traffic in both directions with a single policy entry. For example, entries that contain the patterns `laddr host1` and `raddr host2`, protect traffic in both directions if no direction is specified for the named host. Thus, you need only one policy entry for each host. Policy entries with a format of source address to destination address protect traffic in only one direction. For example, a policy entry of the pattern `saddr host1`

daddr host2 protects inbound traffic or outbound traffic, not both directions. Thus, to protect traffic in both directions, you need to pass the ipsecconf command another entry, as in saddr host2 daddr host1.

You can see the policies that are configured in the system when you issue the ipsecconf command without any arguments. The command displays each entry with an *index* followed by a number. You can use the -d option with the index to delete a particular policy in the system. The command displays the entries in the order that the entries were added, which is not necessarily the order in which the traffic match occurs. To view the order in which the traffic match occurs, use the -l option.

The ipsecpolicy.conf file is deleted when the system shuts down. To ensure that the IPsec policy is active when the machine boots, you can create an IPsec policy file, /etc/inet/ipsecinit.conf, that the inetinit script reads during startup.

IPsec Policy File

To invoke IPsec security policies when you start the Solaris operating environment, you create a configuration file to initialize IPsec with your specific IPsec policy entries. You should name the file /etc/inet/ipsecinit.conf. See the ipsecconf(1M) man page for details about policy entries and their format. After policies are configured, you can use the ipsecconf command to delete a policy temporarily, or to view the existing configuration.

Example—ipsecinit.conf File

The Solaris software includes an IPsec policy file as a sample. This sample file is named ipsecinit.sample. You can use the file as a template to create your own ipsecinit.conf file. The ipsecinit.sample file contains the following examples:

```
#
# For example,
#
#     {rport 23} ipsec {encr_algs des encr_auth_algs md5}
#
# will protect the telnet traffic originating from the host with ESP using
# DES and MD5. Also:
#
#     {raddr 10.5.5.0/24} ipsec {auth_algs any}
#
# will protect traffic to or from the 10.5.5.0 subnet with AH
# using any available algorithm.
#
#
# To do basic filtering, a drop rule may be used. For example:
#
#    {lport 23 dir in} drop {}
```

```
#    {lport 23 dir out} drop {}
#
# will disallow any remote system from telnetting in.
```

Security Considerations for `ipsecinit.conf` and `ipsecconf`

If, for example, the `/etc/inet/ipsecinit.conf` file is sent from an NFS-mounted file system, an adversary can modify the data contained in the file. The outcome would be a change to the configured policy. Consequently, you should use extreme caution if transmitting a copy of the `ipsecinit.conf` file over a network.

The policy cannot be changed for TCP sockets or UDP sockets on which a `connect()` or `accept()` function call has been issued. A socket whose policy cannot be changed is called a latched socket. New policy entries do not protect sockets that are already latched. See the `connect`(3SOCKET) and `accept`(3SOCKET) man pages.

Ensure that you set up the policies before starting any communications, because existing connections might be affected by the addition of new policy entries. Similarly, do not change policies in the middle of a communication.

Protect your naming system. If the following two conditions are met, then your host names are no longer trustworthy:

- Your source address is a host that can be looked up over the network
- Your naming system is compromised

Security weaknesses often lie in misapplication of tools, not the actual tools. You should be cautious when using the `ipsecconf` command. Use a console or other hard-connected TTY for the safest mode of operation.

Security Associations Database for IPsec

Information on keying material for IPsec security services is maintained in a security association database (SADB). Security associations protect both inbound packets and outbound packets. A user process, or possibly multiple cooperating processes, maintains SADBs by sending messages over a special kind of socket. This method of maintaining SADBs is analogous to the method that is described in the `route`(7P) man page. Only a superuser or someone who has assumed an equivalent role can access an SADB.

The operating system might spontaneously emit messages in response to external events. For example, the system might request for a new SA for an outbound datagram, or the system might report the expiration of an existing SA. You open the channel for passing SADB control messages by using the socket call that is mentioned in the previous section. More than one key socket can be open per system.

Messages include a small base header, followed by a number of extension messages. The number of messages might be zero or more. Some messages require additional data. The base message and all extensions must be 8-byte aligned. The GET message serves as an example. This message requires the base header, the SA extension, and the ADDRESS_DST extension. See the `pf_key`(7P) man page for details.

Keying Utilities

The IKE protocol is the automatic keying utility for IPv4 and IPv6 addresses. See Chapter 4 for how to set up IKE. The manual keying utility is the `ipseckey` command. See the `ipseckey`(1M) man page.

You use the `ipseckey` command to manually manipulate the security association databases with the `ipsecah` and `ipsecesp` protection mechanisms. You can also use the `ipseckey` command to set up security associations between communicating parties when automated key management is not used.

While the `ipseckey` command has only a limited number of general options, the command supports a rich command language. You can specify that requests should be delivered by means of a programmatic interface specific for manual keying. See the `pf_key`(7P) man page for additional information. When you invoke the `ipseckey` command with no arguments, the command enters an interactive mode that displays a prompt that enables you to make entries. Some commands require an explicit security association (SA) type, while others permit you to specify the SA type and act on all SA types.

Security Considerations for `ipseckey`

The `ipseckey` command enables a privileged user to enter sensitive cryptographic keying information. If an adversary gains access to this information, the adversary can compromise the security of IPsec traffic. You should consider the following issues when you handle keying material and use the `ipseckey` command:

1. Have you refreshed the keying material? Periodic key refreshment is a fundamental security practice. Key refreshment guards against potential weaknesses of the algorithm and keys, and limits the damage of an exposed key.
2. Is the TTY going over a network? Is the `ipseckey` command in interactive mode?
   - In interactive mode, the security of the keying material is the security of the network path for this TTY's traffic. You should avoid using the `ipseckey` command over a clear-text telnet or rlogin session.
   - Even local windows might be vulnerable to attacks by a concealed program that reads window events.
3. Is the file being accessed over the network? Can the file be read by the world? Have you used the `-f` option?

- An adversary can read a network-mounted file as the file is being read. You should avoid using a world-readable file that contains keying material.
- Protect your naming system. If the following two conditions are met, then your host names are no longer trustworthy:
  - Your source address is a host that can be looked up over the network
  - Your naming system is compromised

Security weaknesses often lie in misapplication of tools, not the actual tools. You should be cautious when using the `ipseckey` command. Use a console or other hard-connected TTY for the safest mode of operation.

IPsec Extensions to Other Utilities

The `ifconfig` command has options to manage the IPsec policy on a tunnel interface. The `snoop` command can parse AH and ESP headers.

`ifconfig` Command

To support IPsec, the following security options have been added to the `ifconfig` command:

- `auth_algs`
- `encr_auth_algs`
- `encr_algs`

You must specify all IPsec security options for a tunnel in one invocation. For example, if you are using only ESP to protect traffic, you would configure the tunnel, `ip.tun0`, once with both security options, as in:

```
# ifconfig ip.tun0 ... encr_algs 3DES encr_auth_algs MD5
```

Similarly, an `ipsecinit.conf` entry would configure the tunnel once with both security options, as in:

```
# WAN traffic uses ESP with 3DES and MD5.
   {} ipsec {encr_algs 3des encr_auth_algs md5}
```

`auth_algs` Security Option

This option enables IPsec AH for a tunnel with a specified authentication algorithm. The `auth_algs` option has the following format:

`auth_algs` *authentication-algorithm*

For the algorithm, you can specify either a number or an algorithm name, including the parameter *any*, to express no specific algorithm preference. To disable tunnel security, specify the following option:

```
auth_algs none
```

See Table 1–1 for a list of available authentication algorithms and for pointers to the algorithm man pages.

`encr_auth_algs` Security Option

This option enables IPsec ESP for a tunnel with a specified authentication algorithm. The `encr_auth_algs` option has the following format:

`encr_auth_algs` *authentication-algorithm*

For the algorithm, you can specify either a number or an algorithm name, including the parameter *any*, to express no specific algorithm preference. If you specify an ESP encryption algorithm, but you do not specify the authentication algorithm, the ESP authentication algorithm value defaults to the parameter *any*.

See Table 1–1 for a list of available authentication algorithms and for pointers to the algorithm man pages.

`encr_algs` Security Option

This option enables IPsec ESP for a tunnel with a specified encryption algorithm. The `encr_algs` option has the following format:

`encr_algs` *encryption-algorithm*

For the algorithm, you can specify either a number or an algorithm name. To disable tunnel security, specify the following option:

```
encr_algs none
```

If you specify an ESP authentication algorithm, but not an encryption algorithm, ESP's encryption value defaults to the parameter *null*.

For a list of available encryption algorithms and for pointers to the algorithm man pages, see the `ipsecesp`(7P) man page or Table 1–2.

`snoop` Command

The `snoop` command can now parse AH and ESP headers. Because ESP encrypts its data, the `snoop` command cannot see encrypted headers that are protected by ESP. AH does not encrypt data, so traffic can still be inspected with this command. The `snoop -V` option shows when AH is in use on a packet. See the `snoop`(1M) man page for more details.

For a sample of verbose `snoop` output on a protected packet, see "How to Verify That Packets Are Protected" on page 47.

CHAPTER 2

Administering IPsec (Tasks)

This chapter provides procedures for implementing IPsec on your network. The procedures are described in "Implementing IPsec (Task Map)" on page 29.

For overview information about IPsec, see Chapter 1. The `ipsecconf`(1M), `ipseckey`(1M), and `ifconfig`(1M) man pages also describe useful procedures in their respective Examples sections.

Implementing IPsec (Task Map)

Task	Description	For Instructions
Secure traffic between two systems	Involves: ■ Adding addresses to the `/etc/inet/ipnodes` file ■ Entering the IPsec policy in the `/etc/inet/ipsecinit.conf` file ■ Setting up key exchange ■ Activating the `ipsecinit.conf` file	"How to Secure Traffic Between Two Systems" on page 31
Secure a web server by using IPsec policy	Involves enabling only secure traffic by entering different security requirements for different ports in the `ipsecinit.conf` file. Also involves activating the file.	"How to Secure a Web Server" on page 33

Task	Description	For Instructions
Set up a virtual private network (VPN)	Involves: ■ Turning off IP forwarding ■ Turning on IP strict destination multihoming ■ Disabling most network and Internet services ■ Adding security associations ■ Configuring the IPsec policy ■ Configuring a secure tunnel ■ Turning on IP forwarding ■ Configuring a default route ■ Running the routing protocol	"How to Set Up a Virtual Private Network (VPN)" on page 35
Generate random numbers	Involves using the `od` command to generate random numbers for keying material when you manually create SAs.	"How to Generate Random Numbers" on page 42
Create or replace security associations (SAs) manually	Involves using the `ipseckey` command to create SAs. Also involves creating an `ipseckeys` file to hold the keying material.	"How to Manually Create IPsec Security Associations" on page 43
Check that IPsec is protecting the packets	Involves examining `snoop` output for specific headers that indicate how the IP datagrams are protected.	"How to Verify That Packets Are Protected" on page 47

IPsec Tasks

This section provides procedures that enable you to secure traffic between two systems, secure a web server, and set up a virtual private network (VPN). Additional procedures provide keying material, provide SAs, and check that IPsec is working as configured.

Some examples in these procedures use the system names `enigma` and `partym`. Substitute the names of your systems for the names `enigma` and `partym`.

For information on how to use roles to administer IPsec, see "Role-Based Access Control (Tasks)" in *System Administration Guide: Security Services*.

▼ How to Secure Traffic Between Two Systems

This procedure assumes the following setup:

- The two systems are named `enigma` and `partym`. Substitute the names of your systems when following this procedure.
- Each system has two addresses, an IPv4 address and an IPv6 address.
- Each system invokes AH protection with the MD5 algorithm, which requires a key of 128 bits.
- Each system invokes ESP protections with the 3DES algorithm, which requires a key of 192 bits.
- IPsec uses shared security associations (SAs).

  With shared SAs, only one pair of SAs is needed to protect the two systems.

1. **On the system console, become superuser or assume an equivalent role.**

Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

---

2. **On each system, add the addresses and host name for the other system in the `/etc/inet/ipnodes` file. The entries for one system must be contiguous in the file.**

   If you are connecting systems with IPv4 addresses only, you modify the `/etc/inet/hosts` file.

   a. **On a system that is named `partym`, type the following in the `ipnodes` file:**

   ```
   # Secure communication with enigma
   192.168.116.16 enigma
   fec0::10:20ff:fea0:21f6 enigma
   ```

   b. **On a system that is named `enigma`, type the following in the `ipnodes` file:**

   ```
   # Secure communication with partym
   192.168.13.213  partym
   fec0::9:a00:20ff:fe7b:b373 partym
   ```

   This step enables the boot scripts to use the system names without depending on nonexistent naming services.

3. **On each system, create the file `/etc/inet/ipsecinit.conf`.**

   You can copy the file `/etc/inet/ipsecinit.sample` to the file `/etc/inet/ipsecinit.conf`.

4. **Add the IPsec policy entry to the `ipsecinit.conf` file.**

a. On the `enigma` system, add the following policy to the `ipsecinit.conf` file:

```
{laddr enigma raddr partym} ipsec {auth_algs any encr_algs any sa shared}
```

b. On the `partym` system, add the same policy to its `ipsecinit.conf` file:

```
{laddr partym raddr enigma} ipsec {auth_algs any encr_algs any sa shared}
```

For the syntax of IPsec policy entries, see the `ipsecconf`(1M) man page.

5. **On each system, add a pair of IPsec SAs between the two systems.**

   You can configure Internet Key Exchange (IKE) to create the SAs automatically. You can also add the SAs manually.

   Note – You should use IKE unless you have good reason to generate and maintain your keys manually. IKE key management is more secure than manual key management.

   - Configure IKE by following one of the configuration procedures in "Configuring IKE (Task Map)" on page 61. For the syntax of the IKE configuration file, see the `ike.config`(4) man page.
   - If you are going to add the SAs manually, see "How to Manually Create IPsec Security Associations" on page 43.

6. **Reboot each system.**

   ```
   # /usr/sbin/reboot
   ```

7. **Verify that packets are being protected. See "How to Verify That Packets Are Protected" on page 47.**

Example—Securing Traffic Between Systems Without Rebooting

The following example describes how to test that the traffic between two systems is secure. In a production environment, it is safer to reboot than to run the `ipsecconf` command.

Instead of rebooting at Step 5 of "How to Secure Traffic Between Two Systems" on page 31, do one of the following options.

- If you used IKE to create keying material, stop and then restart the `in.iked` daemon.

  ```
  # pkill in.iked
  # /usr/lib/inet/in.iked
  ```

- If you added keys manually, use the `ipseckey` command to add the SAs to the database. Then activate the IPsec policy with the `ipsecconf` command.

```
# ipseckey -f /etc/inet/secret/ipseckeys
# ipsecconf -a /etc/inet/ipsecinit.conf
```

Caution – Read the warning when you execute the `ipsecconf` command. A socket that is already latched, that is, the socket is in use, provides an unsecured back door into the system. For more extensive discussion, see "Security Considerations for `ipsecinit.conf` and `ipsecconf`" on page 24.

▼ How to Secure a Web Server

A secure web server allows web clients to talk to the web service. On a secure web server, traffic that is not web traffic *must* pass security checks. The following procedure includes bypasses for web traffic. In addition, this web server can make nonsecured DNS client requests. All other traffic requires ESP with Blowfish and SHA algorithms. Other traffic also uses a shared SA for outbound traffic. Shared SAs reduce the number of SAs that must be generated.

1. **On the system console, become superuser or assume an equivalent role.**

   Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

2. **Determine which services need to bypass security policy checks.**

   For a web server, these services include TCP ports 80 (HTTP) and 443 (Secure HTTP). If the web server provides DNS name lookups, the server might also need to include port 53 for both TCP and UDP.

3. **Create a file in the `/etc/inet` directory for the web server policy. Give the file a name that indicates its purpose, for example `IPsecWebInitFile`. Type the following lines in this file:**

```
# Web traffic that web server should bypass.
    {lport  80 ulp tcp dir both} bypass {}
    {lport 443 ulp tcp dir both} bypass {}

    # Outbound DNS lookups should also be bypassed.
    {rport 53 dir both} bypass {}

    # Require all other traffic to use ESP with Blowfish and SHA-1.
    # Use a shared SA for outbound traffic, in order to avoid a
    # large supply of security associations.
    {} permit {encr_algs blowfish encr_auth_algs sha}
    {} apply {encr_algs blowfish encr_auth_algs sha sa shared}
```

This configuration enables only secure traffic to access the system, with the bypass exceptions that are described in Step 2.

4. **Read the file that you created in Step 3 into the `/etc/inet/ipsecinit.conf` file.**

```
# vi  /etc/inet/ipsecinit.conf
 :r IPsecWebInitFile
 :wq!
```

5. **Protect the `IPsecWebInitFile` file with read-only permissions.**

```
# chmod 400 IPsecWebInitFile
```

6. **Secure the web server without rebooting. Choose one of the following options.**

   - If you are using IKE for key management, stop and restart the `in.iked` daemon.

```
# pkill in.iked
# /usr/lib/inet/in.iked
```

   - If you are manually managing keys, use the `ipseckey` and `ipsecconf` commands.

```
# ipseckey -f /etc/inet/secret/ipseckeys
# ipsecconf -a /etc/inet/IPsecWebInitFile
```

     If you have existing entries in the `ipsecinit.conf` file, the file generates errors when read again.

Caution – Read the warning when you execute the `ipsecconf` command. A socket that is already latched, that is, the socket is in use, provides an unsecured back door into the system. For more extensive discussion, see "Security Considerations for `ipsecinit.conf` and `ipsecconf`" on page 24. The same warning applies to restarting the `in.iked` daemon.

You can also reboot. Rebooting ensures that the IPsec policy is in effect on all TCP connections. At reboot, the TCP connections use the policy in the IPsec policy file.

The web server now allows only web-server traffic, as well as outbound DNS requests and replies. No other services work without enabling IPsec on a remote system.

7. **(Optional) To enable a remote system to communicate with the web server for nonweb traffic, put the following policy in a remote system's `ipsecinit.conf` file.**

   A remote system can communicate securely with the web server for nonweb traffic only when their IPsec policies match.

```
# Communicate with web server about nonweb stuff
#
{saddr webserver} permit {encr_algs blowfish encr_auth_algs sha}
{saddr webserver} apply {encr_algs blowfish encr_auth_algs sha sa shared}
```

▼ How to Set Up a Virtual Private Network (VPN)

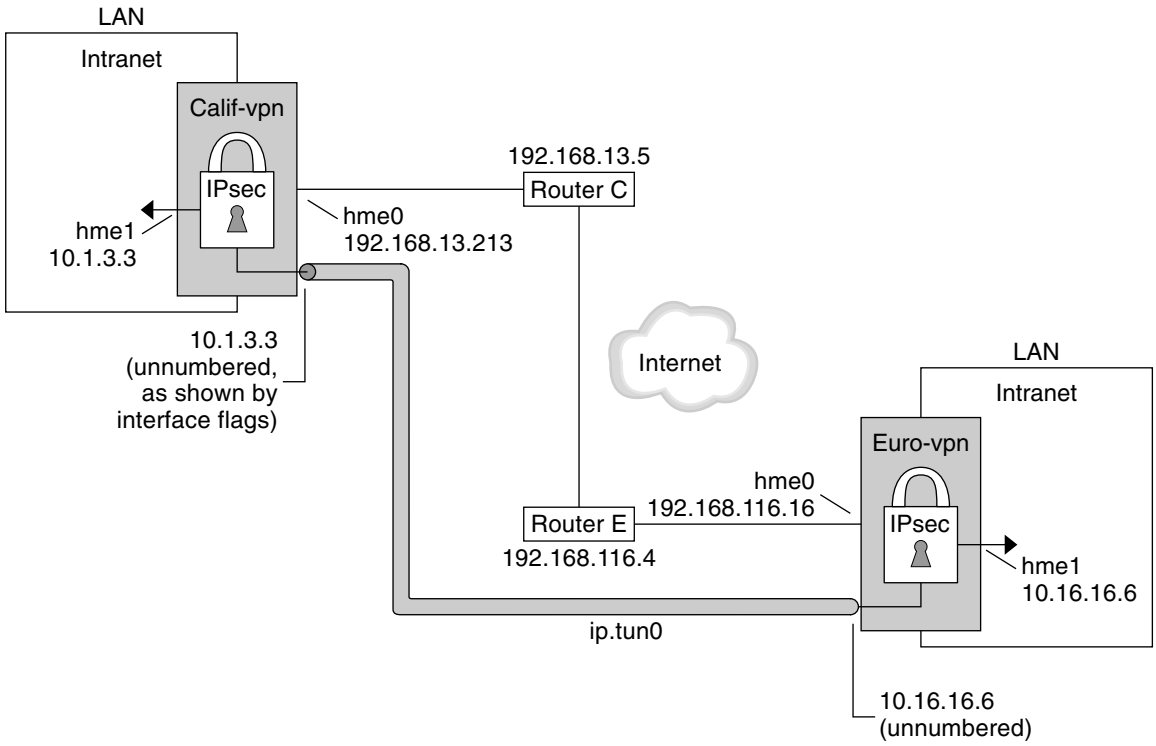
This procedure shows you how to set up a VPN by using the Internet to connect two networks within an organization. The procedure then shows you how to secure the traffic between the networks with IPsec.

This procedure extends the procedure, "How to Secure Traffic Between Two Systems" on page 31. In addition to connecting two systems, you are connecting two intranets that connect to these two systems. The systems in this procedure function as gateways.

This procedure assumes the following setup:

- Each system is using an IPv4 address space. This procedure also works with IPv6 addresses.
- Each system has two interfaces. The `hme0` interface connects to the Internet. In this example, Internet IP addresses begin with `192.168`. The `hme1` interface connects to the company's local area network (LAN), its intranet. In this example, intranet IP addresses begin with the number `10`.
- Each system invokes AH protection with the MD5 algorithm. The MD5 algorithm requires a 128-bit key.
- Each system invokes ESP protection with the 3DES algorithm. The 3DES algorithm requires a 192-bit key.
- Each system can connect to a router that has direct access to the Internet.
- IPsec uses shared security associations (SAs).

For a description of VPNs, see "Virtual Private Networks" on page 21. The following figure describes the VPN that this procedure configures.



hme0 = Turn off IP forwarding

hme1 = Turn on IP forwarding

ip.tun = Turn on IP forwarding

Router C — /etc/defaultrouter for Calif-vpn

Router E — /etc/defaultrouter for Euro-vpn

This procedure uses the following configuration parameters.

Parameter	Europe	California
System name	enigma	partym
System intranet interface	hme1	hme1
System Internet interface	hme0	hme0
System intranet address, also the *-point* address in Step 8	10.16.16.6	10.1.3.3

Parameter	Europe	California
System Internet address, also the *-taddr* address in Step 8	192.168.116.16	192.168.13.213
Name of Internet router	router-E	router-C
Address of Internet router	192.168.116.4	192.168.13.5
Tunnel name	`ip.tun0`	`ip.tun0`

1. **On the system console on one of the systems, become superuser or assume an equivalent role.**

   Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

2. **Turn off IP forwarding. Choose one of the following options:**

   - On IPv4 networks, use this syntax:

     ```
     # ndd -set /dev/ip ip_forwarding 0
     ```

   - On IPv6 networks, use this syntax:

     ```
     # ndd -set /dev/ip6 ip6_forwarding 0
     ```

   Turning off IP forwarding prevents packets from being forwarded from one network to another network through this system. For a description of the `ndd` command, see the `ndd`(1M) man page.

3. **Turn on IP strict destination multihoming. Choose one of the following options:**

   - On IPv4 networks, use this syntax:

     ```
     # ndd -set /dev/ip ip_strict_dst_multihoming 1
     ```

   - On IPv6 networks, use this syntax:

     ```
     # ndd -set /dev/ip6 ip6_strict_dst_multihoming 1
     ```

   Turning on IP strict destination multihoming ensures that packets for one of the system's destination addresses arrive at the correct destination address.

   When you use the `ndd` command to turn off IP forwarding and turn on IP strict destination multihoming, fewer packets flow all the way through the system. When strict destination multihoming is enabled, packets that arrive on a particular interface must be addressed to one of the local IP addresses of that interface. All other packets, even ones addressed to other local addresses of the system, are dropped.

4. **Disable most network services, and possibly all network services, by doing the following substeps, as needed:**

a. **Edit the `inetd.conf` file to remove all but essential services. Then, force the `inetd` daemon to reread the `inetd.conf` file.**

```
# pkill -HUP inetd
```

Note – The VPN router should allow very few incoming requests. You need to disable all processes that accept incoming traffic. For example, you might comment out lines in the `inetd.conf` file, you might kill SNMP, and so on. Alternatively, you can use techniques that are similar to the techniques in "How to Secure a Web Server" on page 33.

---

b. **If the `inetd.conf` file has not been edited to remove all but essential services, kill the the `inetd` daemon.**

```
# pkill inetd
```

c. **Disable other Internet services, such as SNMP, NFS, and so on, by typing the appropriate commands. For example, the following commands kill the NFS service and the mail service:**

```
# /etc/init.d/nfs.server stop
# /etc/init.d/sendmail stop
```

The disabling of network services prevents IP packets from doing any harm to the system. For example, an SNMP daemon, a `telnet` connection, or an `rlogin` connection could be exploited.

5. **On each system, add a pair of SAs between the two systems.**

Configure IKE to manage the keys for the SAs. Use one of the procedures in "Configuring IKE (Task Map)" on page 61 to configure IKE for the VPN.

If you have an overriding reason to manually manage the keys, see "How to Manually Create IPsec Security Associations" on page 43.

6. **On each system, edit the `/etc/inet/ipsecinit.conf` file to add the VPN policy.**

a. **For example, on the `enigma` system, type the following entries into the `ipsecinit.conf` file:**

```
# LAN traffic can bypass IPsec.
   {laddr 10.16.16.6 dir both} bypass {}

# WAN traffic uses ESP with 3DES and MD5.
   {} ipsec {encr_algs 3des encr_auth_algs md5}
```

b. **On the `partym` system, type the following entries into the `ipsecinit.conf` file:**

```
# LAN traffic can bypass IPsec.
   {laddr 10.1.3.3 dir both} bypass {}
```

```
# WAN traffic uses ESP with 3DES and MD5.
   {} ipsec {encr_algs 3des encr_auth_algs md5}
```

The `ipsec` entry prevents remote systems from sending clear packets. The `bypass` entry allows nodes that are part of the LAN to treat the VPN router as if the router were part of the LAN.

7. **(Optional) For a higher level of security, remove the LAN `bypass` entry.**

   The entry in the `ipsecinit.conf` file would appear similar to the following:

   ```
   # All traffic uses ESP with 3DES and MD5.
      {} ipsec {encr_algs 3des encr_auth_algs md5}
   ```

   Each system on the LAN would then need to activate IPsec to communicate with the VPN router.

8. **On each system, configure a secure tunnel, `ip.tun0`.**

   The tunnel adds another physical interface from the IP perspective. Type the following three `ifconfig` commands to create the point-to-point interface:

   ```
   # ifconfig ip.tun0 plumb

   # ifconfig ip.tun0 system1-point system2-point \
   tsrc system1-taddr tdst system2-taddr encr_algs 3DES encr_auth_algs MD5

   # ifconfig ip.tun0 up
   ```

   a. **For example, on the `enigma` system, type the following commands:**

      ```
      # ifconfig ip.tun0 plumb

      # ifconfig ip.tun0 10.16.16.6 10.1.3.3   \
      tsrc 192.168.116.16 tdst 192.168.13.213 \
      encr_algs 3DES encr_auth_algs MD5

      # ifconfig ip.tun0 up
      ```

   b. **On the `partym` system, type the following commands:**

      ```
      # ifconfig ip.tun0 plumb

      # ifconfig ip.tun0 10.1.3.3 10.16.16.6  \
      tsrc 192.168.13.213 tdst 192.168.116.16 \
      encr_algs 3DES encr_auth_algs MD5

      # ifconfig ip.tun0 up
      ```

   The policy that is passed to the `ifconfig` commands must be the same as the policy in the `ipsecinit.conf` file. Upon reboot, each system uses the policy in its `ipsecinit.conf` file.

9. **On each system, turn on IP forwarding for the `hme1` and `ip.tun0` interfaces. Choose one of the following options:**

   - On IPv4 networks, use this syntax:

```
# ndd -set /dev/ip hme1:ip_forwarding 1
# ndd -set /dev/ip ip.tun0:ip_forwarding 1
```

- On IPv6 networks, use this syntax:

```
# ndd -set /dev/ip6 hme1:ip6_forwarding 1
# ndd -set /dev/ip6 ip.tun0:ip6_forwarding 1
```

IP forwarding means that packets that arrive from somewhere else can be forwarded. IP forwarding also means that packets that leave this interface might have originated somewhere else. To successfully forward a packet, both the receiving interface and the transmitting interface must have IP forwarding turned on.

Because the `hme1` interface is *inside* the intranet, IP forwarding must be turned on for `hme1`. Because `ip.tun0` connects the two systems through the Internet, IP forwarding must be turned on for `ip.tun0`.

The `hme0` interface has its IP forwarding turned off to prevent an *outside* adversary from injecting packets into the protected intranet. The *outside* refers to the Internet.

10. On each system, ensure that routing protocols do not advertise the default route within the intranet.

```
# ifconfig hme0 private
```

Even if `hme0` has IP forwarding turned off, a routing protocol implementation might still advertise the interface. For example, the `in.routed` protocol might still advertise that `hme0` is available to forward packets to its peers inside the intranet. Setting the interface's *private* flag prevents these advertisements.

11. Manually, add a default route over `hme0`.

The default route should be a router with direct access to the Internet.

```
# pkill in.rdisc
# route add default router-on-hme0-subnet
```

a. For example, on the `enigma` system, add the following route:

```
# pkill in.rdisc
# route add default 192.168.116.4
```

b. On the `partym` system, add the following route:

```
# pkill in.rdisc
# route add default 192.168.13.5
```

Even though the `hme0` interface is not part of the intranet, `hme0` does need to reach across the Internet to its peer system. To find its peer, `hme0` needs information about Internet routing. The VPN system appears to be a host, rather than a router, to the rest of the Internet. Therefore, you can use a default router or run the router discovery protocol to find a peer system. For more information, see the `route`(1M) and `in.routed`(1M) man pages.

12. Ensure that `hme0` uses the default route after a reboot by creating a `defaultrouter` file.

Put the IP address of `hme0`'s default router in the `/etc/defaultrouter` file. This step prevents the `in.rdisc` daemon from being started at reboot.

a. For example, on the `enigma` system, add the Internet router for `enigma` in the `/etc/defaultrouter` file.

```
# vi /etc/defaultrouter

192.168.116.4 router-E
```

b. Add the `partym` system's Internet router in `partym`'s `/etc/defaultrouter` file.

```
# vi /etc/defaultrouter

192.168.13.5 router-C
```

13. On each system, prevent routing from occurring early in the boot sequence, thus reducing vulnerability.

```
# touch /etc/notrouter
```

14. Ensure that the VPN starts after a reboot by editing the `/etc/hostname.ip.tun0` file.

system1-point system2-point `tsrc` *system1-taddr* `\`
`tdst` *system2-taddr* `encr_algs 3des encr_auth_algs md5 up`

a. For example, on the `enigma` system, add the following lines to the `hostname.ip.tun0` file:

```
10.16.16.6 10.1.3.3 tsrc 192.168.116.16 \
tdst 192.168.13.213 encr_algs 3DES encr_auth_algs MD5 up
```

b. On the `partym` system, add the following lines to the `hostname.ip.tun0` file:

```
10.1.3.3 10.16.16.6 tsrc 192.168.13.213 \
tdst 192.168.116.16 encr_algs 3DES encr_auth_algs MD5 up
```

15. On each system, create a file that configures some VPN parameters at boot time. Name the file `/etc/rc3.d/S99vpn_setup`.

On each system, turn on IP forwarding for the `hme1` and `ip.tun0` interfaces. Choose one of the following options:

- On an IPv4 network, type the following lines in the file:

```
ndd -set /dev/ip hme1:ip_forwarding 1
ndd -set /dev/ip ip.tun0:ip_forwarding 1
ifconfig hme0 private
in.routed
```

- On an IPv6 network, type the following lines in the file:

```
ndd -set /dev/ip6 hme1:ip6_forwarding 1
ndd -set /dev/ip6 ip.tun0:ip6_forwarding 1
ifconfig hme0 private
in.routed
```

You can also manually add routes in the `/etc/rc3.d/S99vpn_setup` file, instead of using the `in.routed` protocol.

16. On each system, run a routing protocol.

```
# in.routed
```

▼ How to Generate Random Numbers

If you are entering keys manually, the keying material should be random. The format for keying material is hexadecimal.

If your site has a random number generator, use that generator. Otherwise, you can use the `od` command with the `/dev/random` Solaris device as input. For more information, see the `od`(1) man page.

1. Generate random numbers in hexadecimal format.

`% od -x|-X -A n` *file*

`-x` Displays the octal dump in hexadecimal format. Hexadecimal format is useful for keying material. The hexadecimal is printed in 4-character chunks.

`-X` Displays the octal dump in hexadecimal format. The hexadecimal is printed in 8-character chunks.

`-A n` Removes the input offset base from the display.

file Serves as a source for random numbers.

For example, the following commands print hexadecimal numbers.

```
% od -X -A n /dev/random | head -2
         d54d1536 4a3e0352 0faf93bd 24fd6cad
         8ecc2670 f3447465 20db0b0c c83f5a4b
% od -x -A n /dev/random | head -2
         34ce 56b2 8b1b 3677 9231 42e9 80b0 c673
         2f74 2817 8026 df68 12f4 905a db3d ef27
```

2. Combine the output to create a key of the appropriate length.

Remove the spaces between the numbers on one line to create a 32-character key. A 32-character key is 128 bits. For a security parameter index (SPI), you can use an 8-character key.

▼ How to Manually Create IPsec Security Associations

Although manual management of IPsec security associations (SAs) is not recommended for security reasons, you can do so. The following procedure works with the procedure, "How to Secure Traffic Between Two Systems" on page 31. You first create SAs with the `ipseckey` command. You then place the keying material in the `ipseckeys` file.

1. **Generate the keying material for the SAs.**

   You need three hexadecimal random numbers for outbound traffic, and three hexadecimal random numbers for inbound traffic. Therefore, one system needs to generate the following numbers:

   - Two hexadecimal random numbers as the value for the `spi` keyword. One number is for outbound traffic. One number is for inbound traffic. Each number can be up to eight characters long.
   - Two hexadecimal random numbers for the MD5 algorithm for AH. Each number must be 32 characters long. One number is for `dst enigma`. One number is for `dst partym`.
   - Two hexadecimal random numbers for the 3DES algorithm for ESP. For a 192-bit key, each number must be 48 characters long. One number is for `dst enigma`. One number is for `dst partym`.

   If you have a random number generator at your site, use the generator. You can also use the `od` command. See "How to Generate Random Numbers" on page 42 for the procedure.

2. **On the system console on one of the systems, become superuser or assume an equivalent role.**

   Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

   ---

3. **Enable the `ipseckey` command mode:**

   ```
   # ipseckey

   >
   ```

   The > prompt indicates that you are in `ipseckey` command mode.

4. **If you are replacing existing SAs, flush the current SAs.**

   ```
   > flush
   >
   ```

To prevent an adversary from having time to break your SAs, you need to replace the keying material.

Note – You must coordinate key replacement on communicating systems. When you replace the SAs on one system, the SAs must also be replaced on the remote system.

---

5. **To create SAs, type the following command.**

   You also use this syntax to replace SAs that you have just flushed.

   ```
   > add protocol spi random-hex-string \
   src addr dst addr2 \
   protocol-prefix_alg protocol-algorithm \
   protocol-prefixkey random-hex-string-of-algorithm-specified-length
   ```

   random-hex-string
   Specifies a random number of up to eight characters in hexadecimal format. Precede the characters with `0x`. If you enter more numbers than the security parameter index (SPI) accepts, the system ignores the extra numbers. If you enter fewer numbers than the SPI accepts, the system pads your entry.

   protocol
   Specifies either `esp` or `ah`.

   addr
   Specifies the IP address of one system.

   addr2
   Specifies the IP address of the peer system of *addr*.

   protocol-prefix
   Specifies one of `encr` or `auth`. The `encr` prefix is used with the `esp` protocol. The `auth` prefix is used with the `ah` protocol. The option `encr_auth_alg` is used with the `esp` protocol.

   protocol-algorithm
   Specifies an algorithm for ESP or AH. Each algorithm requires a key of a specific length.

   Authentication algorithms include MD5 and SHA. Encryption algorithms include 3DES and AES.

   random-hex-string-of-algorithm-specified-length
   Specifies a random hexadecimal number of the length that is required by the algorithm. For example, the MD5 algorithm requires a 32-character string for its 128-bit key. The 3DES algorithm requires a 48-character string for its 192-bit key.

   a. **For example, on the `enigma` system, type the following commands to protect outbound packets. Use the random numbers that you generated in Step 1.**

```
> add esp spi 0x8bcd1407 \
src 192.168.116.16 dst 192.168.13.213 \
encr_alg 3DES \
encrkey d41fb74470271826a8e7a80d343cc5aae9e2a7f05f13730d

> add ah spi 0x18907dae \
src 192.168.116.16 dst 192.168.13.213 \
auth_alg MD5 \
authkey e896f8df7f78d6cab36c94ccf293f031

>
```

Note – The peer system must use the same keying material.

---

b. Still in `ipseckey` command mode on the `enigma` system, type the following commands to protect inbound packets:

```
> add esp spi 0x122a43e4 \
src 192.168.13.213 dst 192.168.116.16 \
encr_alg 3des \
encrkey dd325c5c137fb4739a55c9b3a1747baa06359826a5e4358e

> add ah spi 0x91825a77 \
src 192.168.13.213 dst 192.168.116.16 \
auth_alg md5 \
authkey ad9ced7ad5f255c9a8605fba5eb4d2fd

>
```

Note – The keys and SPI can be different for each SA. You *should* assign different keys and a different SPI for each SA.

---

6. To exit `ipseckey` command mode, press Control-D or type `quit`.

7. To ensure that the keying material is available to IPsec at reboot, add the keying material to the `/etc/inet/secret/ipseckeys` file.

The lines of the `/etc/inet/secret/ipseckeys` file are identical to the command line language.

a. For example, the `/etc/inet/secret/ipseckeys` file on the `enigma` system would appear similar to the following:

```
# ipseckeys - This file takes the file format documented in
#   ipseckey(1m).
#   Note that naming services might not be available when this file
#   loads, just like ipsecinit.conf.
#
# for outbound packets on enigma
```

```
add esp spi 0x8bcd1407 \
   src 192.168.116.16 dst 192.168.13.213  \
   encr_alg 3DES \
   encrkey  d41fb74470271826a8e7a80d343cc5aae9e2a7f05f13730d
#
add ah spi  0x18907dae \
   src 192.168.116.16 dst 192.168.13.213 \
   auth_alg MD5  \
   authkey  e896f8df7f78d6cab36c94ccf293f031
#
# for inbound packets
add esp spi 0x122a43e4 \
   src 192.168.13.213 dst 192.168.116.16 \
   encr_alg 3DES \
   encrkey dd325c5c137fb4739a55c9b3a1747baa06359826a5e4358e
#
add ah spi  0x91825a77 \
   src 192.168.13.213 dst 192.168.116.16 \
   auth_alg MD5  \
   authkey  ad9ced7ad5f255c9a8605fba5eb4d2fd
```

b. Protect the file with read-only permissions.

```
# chmod 400 /etc/inet/secret/ipseckeys
```

8. **Repeat Step 2 through Step 7 on the `partym` system. Use the same keying material that was used on `enigma`.**

   The keying material on the two systems must be identical. As shown in the following example, only the comments in the `ipseckeys` file differ. The comments differ because `dst enigma` is inbound on the `enigma` system, and outbound on the `partym` system.

```
# partym ipseckeys file
#
#for inbound packets
add esp spi 0x8bcd1407 \
   src 192.168.116.16 dst 192.168.13.213  \
   encr_alg 3DES \
   encrkey  d41fb74470271826a8e7a80d343cc5aae9e2a7f05f13730d
#
add ah spi  0x18907dae \
   src 192.168.116.16 dst 192.168.13.213 \
   auth_alg MD5  \
   authkey  e896f8df7f78d6cab36c94ccf293f031
#
# for outbound packets
add esp spi 0x122a43e4 \
   src 192.168.13.213 dst 192.168.116.16 \
   encr_alg 3DES \
   encrkey dd325c5c137fb4739a55c9b3a1747baa06359826a5e4358e
#
add ah spi  0x91825a77 \
   src 192.168.13.213 dst 192.168.116.16 \
   auth_alg MD5  \
```

```
authkey  ad9ced7ad5f255c9a8605fba5eb4d2fd
```

▼ How to Verify That Packets Are Protected

To verify that packets are protected, test the connection with the `snoop` command. The following prefixes can appear in the `snoop` output:

- `AH:` Prefix indicates that AH is protecting the headers. You see `AH:` if you used `auth_alg` to protect the traffic.
- `ESP:` Prefix indicates that encrypted data is being sent. You see `ESP:` if you used `encr_auth_alg` or `encr_alg` to protect the traffic.

Note – You must be superuser or have assumed an equivalent role to read the `snoop` output. You must have access to both systems to test the connection.

---

1. **On one system, such as `partym`, become superuser.**

```
% su
Password:       Type root password
#
```

2. **In a terminal window, begin to snoop the packets from a remote system, such as the `enigma` system.**

```
# snoop -v enigma
Using device /dev/hme (promiscuous mode)
```

3. **In another terminal window, remotely log in to the `enigma` system. Provide your password. Then, become superuser, and send a packet from the `enigma` system to the `partym` system.**

```
% rlogin enigma
Password:       Type your password
% su
Password:       Type root password
# ping partym
```

4. **In the `snoop` window on the `partym` system, you should see output that is similar to the following:**

```
IP:   Time to live = 64 seconds/hops
IP:   Protocol = 51 (AH)
IP:   Header checksum = 4e0e
IP:   Source address = 192.168.116.16, enigma
IP:   Destination address = 192.168.13.213, partym
IP:   No options
IP:
AH:  ----- Authentication Header -----
```

```
AH:
AH:  Next header = 50 (ESP)
AH:  AH length = 4 (24 bytes)
AH:  <Reserved field = 0x0>
AH:  SPI = 0xb3a8d714
AH:  Replay = 52
AH:  ICV = c653901433ef5a7d77c76eaa
AH:
ESP:  ----- Encapsulating Security Payload -----
ESP:
ESP:  SPI = 0xd4f40a61
ESP:  Replay = 52
ESP:     ....ENCRYPTED DATA....

ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 20 arrived at 9:44:36.59
ETHER:  Packet size = 98 bytes
ETHER:  Destination = 8:0:27:aa:11:11, Sun
ETHER:  Source      = 8:0:22:aa:22:2, Sun
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:         .... ..0. = not ECN capable transport
IP:         .... ...0 = no ECN congestion experienced
IP:   Total length = 84 bytes
IP:   Identification = 40933
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 60 seconds/hops
IP:   Protocol = 51 (AH)
IP:   Header checksum = 22cc
...
```

CHAPTER 3

Internet Key Exchange (Overview)

The management of keying material that IPsec security associations (SAs) require for secure transmission of IP datagrams is called *key management*. Automatic key management requires a secure channel of communication for the creation, authentication, and exchange of keys. The Solaris operating environment uses Internet Key Exchange (IKE) to automate key management. IKE easily scales to provide a secure channel for a large volume of traffic. IPsec SAs on IPv4 and IPv6 packets can take advantage of IKE.

When IKE is used on a system with a Sun™ Crypto Accelerator 1000 board or a Sun Crypto Accelerator 4000 board, the public key operations are off-loaded to the accelerator. Operating system resources are not used for publickey operations. When IKE is used on a system with a Sun Crypto Accelerator 4000 board, the certificates, public keys, and private keys can be stored on the board. Key storage off the system provides an additional layer of protection.

This chapter contains the following information:

For instructions on implementing IKE, see Chapter 4.

IKE Overview

The IKE daemon, `in.iked`, negotiates and authenticates keying material for SAs in a protected manner. The daemon uses random seeds for keys from internal functions provided by the Solaris operating environment. IKE provides perfect forward secrecy (PFS). In PFS, the keys that protect data transmission are not used to derive additional keys. Also, seeds used to create data transmission keys are not reused. See the `in.iked`(1M) man page.

When the IKE daemon discovers a remote system's public encryption key, the local system can then use that key. The system encrypts messages by using the remote system's public key. The messages can be read only by that remote system. The IKE daemon performs its job in two phases. The phases are called *exchanges*.

Phase 1 Exchange

The Phase 1 exchange is known as *Main Mode*. In the Phase 1 exchange, IKE uses public key encryption methods to authenticate itself with peer IKE entities. The result is an ISAKMP (Internet Security Association and Key Management Protocol) security association (SA). An ISAKMP SA is a secure channel for IKE to negotiate keying material for the IP datagrams. Unlike IPsec SAs, the ISAKMP SAs are bidirectional, so only one security association is needed.

How IKE negotiates keying material in the Phase 1 exchange is configurable. IKE reads the configuration information from the `/etc/inet/ike/config` file. Configuration information includes the following:

- Global parameters, such as the names of public key certificates
- If perfect forward secrecy (PFS) is used
- The interfaces that are affected
- The algorithms that are used
- The authentication method

The two authentication methods are preshared keys and public key certificates. The public key certificates can be self-signed, or the certificates can be issued by a certificate authority (CA) from a PKI (public key infrastructure) organization. Organizations include Baltimore Technologies, Entrust, GeoTrust, RSA Security, Sun Open Net Environment (Sun ONE) Certificate Server, and Verisign.

Phase 2 Exchange

The Phase 2 exchange is known as *Quick Mode*. In the Phase 2 exchange, IKE creates and manages the IPsec SAs between systems that are running the IKE daemon. IKE uses the secure channel that was created in the Phase 1 exchange to protect the

transmission of keying material. The IKE daemon creates the keys from a random number generator by using the `/dev/random` device. The daemon refreshes the keys at a configurable rate. The keying material is available to algorithms that are specified in the configuration file for IPsec policy, `ipsecinit.conf`.

IKE Configuration Choices

The `/etc/inet/ike/config` configuration file contains IKE policy entries. For two IKE daemons to authenticate each other, the configuration file must be valid. Also, keying material must be available. The entries in the configuration file determine the method for using the keying material to authenticate the Phase 1 exchange. The choices are preshared keys or public key certificates.

The entry `auth_method preshared` indicates that preshared keys are used. Values for `auth_method` other than `preshared` indicate that public key certificates are to be used. Public key certificates can be self-signed, or the certificates can be installed from a PKI organization. See the `ike.config`(4) man page.

IKE With Preshared Keys

Preshared keys are created by an administrator on one system, and are shared out of band with administrators of remote systems. You should take care to create large random keys and to protect the file and the out-of-band transmission. The keys are placed in the `/etc/inet/secret/ike.preshared` file on each system. The `ike.preshared` file is for IKE as the `ipseckeys` file is for IPsec. Any compromise of the keys in the `ike.preshared` file compromises all keys that are derived from the keys in the file.

One system's preshared key must be identical to its remote system's key. The keys are tied to a particular IP address. They are most secure when one administrator controls the communicating systems. See the `ike.preshared`(4) man page.

IKE With Public Key Certificates

Public key certificates eliminate the need for communicating systems to share secret keying material out of band. Public keys use the Diffie-Hellman protocol for authenticating and negotiating keys. Public key certificates come in two flavors. The certificates can be self-signed, or the certificates can be certified by a certificate authority (CA).

Self-signed public key certificates are created by an administrator. The `ikecert certlocal -ks` command creates the private part of the public-private key pair for the system. The administrator then gets the self-signed certificate output in X.509 format from the remote system. The remote system's certificate is input to the `ikecert certdb` command for the public part of the key pair. The self-signed certificates reside in the `/etc/inet/ike/publickeys` directory on the communicating systems. When you use the `-T` option, the certificates reside on attached hardware.

Self-signed certificates are a halfway point between preshared keys and CAs. Unlike preshared keys, a self-signed certificate can be used on a mobile machine, or on a system that might be renumbered. To self-sign a certificate for a system without a fixed number, use a `DNS` (`www.example.org`) or `EMAIL` (`root@domain.org`) alternative name.

Public keys can be delivered by a PKI or a CA organization. You install the public keys and their accompanying CAs in the `/etc/inet/ike/publickeys` directory. When you use the `-T` option, the certificates reside on attached hardware. Vendors also issue certificate revocation lists (CRLs). Along with installing the keys and CAs, you are responsible for installing the CRLs in the `/etc/inet/ike/crls` directory.

CAs have the advantage of being certified by an outside organization, rather than by the administrator of the site. In a sense, CAs are notarized certificates. As with self-signed certificates, CAs can be used on a mobile machine, or on a system that might be renumbered. Unlike self-signed certificates, CAs can very easily scale to protect a large number of communicating systems.

IKE and Hardware Acceleration

IKE algorithms are computationally expensive, particularly in the Phase 1 exchange. Systems that handle a large number of exchanges can use a Sun Crypto Accelerator 1000 board to handle the public key operations. The Sun Crypto Accelerator 4000 board can also be used to handle expensive Phase 1 computations.

For information on how to configure IKE to off-load its computations to the accelerator board, see "How to Use the Sun Crypto Accelerator 1000 Board With IKE" on page 86. For information on how to store keys, see "How to Use the Sun Crypto Accelerator 4000 Board With IKE" on page 87.

IKE and Hardware Storage

Public key certificates, private keys, and public keys can be stored on a Sun Crypto Accelerator 4000 board. For RSA, the board supports keys up to 2048 bits. For DSA, the board supports keys up to 1024 bits.

For information on how to configure IKE to access the board, see "How to Use the Sun Crypto Accelerator 1000 Board With IKE" on page 86. For information on how to add certificates and public keys to the board, see "How to Generate and Store Public Key Certificates on Hardware" on page 80.

IKE Utilities and Files

This section describes the configuration files for IKE policy, and the various commands that implement IKE. For instructions about how to implement IKE for your network, see "Configuring IKE (Task Map)" on page 61.

TABLE 3–1 IKE Configuration Files and Commands

File or Command	Description
`in.iked` daemon	Internet Key Exchange (IKE) daemon. Activates automated key management.
`ikeadm` command	IKE administration command for viewing and modifying the IKE policy.
`ikecert` command	Certificate database management command for manipulating local publickey certificate databases. The databases can also be stored on an attached Sun Crypto Accelerator 4000 board.
`/etc/inet/ike/config` file	Configuration file for the IKE policy. Contains the site's rules for matching inbound IKE requests and preparing outbound IKE requests. If this file exists, the `in.iked` daemon starts automatically at boot time.
`/etc/inet/secret/ike.preshared` file	Preshared keys file. Contains secret keying material for authentication in the Phase 1 exchange. Used when configuring IKE with preshared keys.

TABLE 3–1 IKE Configuration Files and Commands *(Continued)*

File or Command	Description
`/etc/inet/secret/ike.privatekeys` file	Private keys directory. Contains the private keys that are part of a public-private key pair.
`/etc/inet/ike/publickeys` directory	Directory that holds public keys and certificate files. Contains the public key part of a public-private key pair.
`/etc/inet/ike/crls` directory	Directory that holds revocation lists for public keys and certificate files.
Sun Crypto Accelerator 1000 board	Hardware that accelerates public key operations by off-loading the operations from the operating system.
Sun Crypto Accelerator 4000 board	Hardware that accelerates public key operations by off-loading the operations from the operating system. The board also stores public keys, private keys, and public key certificates.

IKE Daemon

The `in.iked` daemon automates the management of cryptographic keys for IPsec on a Solaris system. The daemon negotiates with a remote system that is running the same protocol to provide authenticated keying materials for security associations in a protected manner. The daemon must be running on all systems that plan to communicate securely.

The IKE daemon is automatically loaded at boot time if the configuration file for the IKE policy, `/etc/inet/ike/config`, exists. The daemon checks the syntax of the configuration file.

When the IKE daemon runs, the system authenticates itself to its peer IKE entity in the Phase 1 exchange. The peer is defined in the IKE policy file, as are the authentication methods. The daemon then establishes the keys for the Phase 2 exchange. At an interval specified in the policy file, the IKE keys are refreshed automatically. The `in.iked` daemon listens for incoming IKE requests from the network and for requests for outbound traffic through the `PF_KEY` socket. See the `pf_key`(7P) man page for more information.

Two commands support the IKE daemon. The `ikeadm` command enables you to view and modify the IKE policy. The `ikecert` command enables you to view and manage the publickey databases. This command manages the local databases, `ike.privatekeys` and `publickeys`. This command also manages public key operations and the storage of public keys on hardware.

IKE Policy File

The configuration file for the IKE policy, `/etc/inet/ike/config`, provides the keying rules and global parameters for the IKE daemon itself, and for the IPsec SAs that the file manages. The IKE daemon itself requires keying material in the Phase 1 exchange. Rules in the `ike/config` file establish the keying material. A valid rule in the policy file contains a label. The rule identifies the systems or networks that the keying material secures, and specifies the authentication method. See "Configuring IKE With Preshared Keys (Task Map)" on page 62 for examples of valid policy files. For examples and descriptions of its entries, see the `ike.config`(4) man page.

The IPsec SAs are used on the IP datagrams that are protected according to policies that are set up in the configuration file for the IPsec policy, `/etc/inet/ipsecinit.conf`. The IKE policy file determines if PFS is used when creating the IPsec SAs.

The `ike/config` file can include the path to a library that is implemented according to the following standard: RSA Security Inc. PKCS #11 Cryptographic Token Interface (Cryptoki). IKE uses this PKCS #11 library to access hardware for key acceleration and key storage.

The security considerations for the `ike/config` file are similar to the considerations for the `ipsecinit.conf` file. See "Security Considerations for `ipsecinit.conf` and `ipsecconf`" on page 24 for details.

IKE Administration Command

You can use the `ikeadm` command to do the following:

- View aspects of the IKE daemon process
- Change the parameters that are passed to the IKE daemon
- Display statistics on SA creation during the Phase 1 exchange
- Debug IKE processes

See the `ikeadm`(1M) man page for examples and a full description of this command's options. The privilege level of the running IKE daemon determines which aspects of the IKE daemon can be viewed and modified. You can choose from three levels of privilege.

`0x0`, or base level	At the base level of privilege, you cannot view or modify keying material. The base level is the default level at which the `in.iked` daemon runs.
`0x1`, or modkeys level	At the modkeys level of privilege, you can remove, change, and add preshared keys.
`0x2`, or keymat level	At the keymat level of privilege, you can view the actual keying material with the `ikeadm` command.

The security considerations for the ikeadm command are similar to the considerations for the ipseckey command. See "Security Considerations for ipseckey" on page 25 for details.

Preshared Keys Files

The /etc/inet/secret directory contains the preshared keys for ISAKMP SAs and IPsec SAs. When you create preshared keys manually, the ike.preshared file contains the preshared keys for ISAKMP SAs, and the ipseckeys file contains the preshared keys for IPsec SAs. The secret directory is protected at 0700. The files in the secret directory are protected at 0600.

- You create an ike.preshared file when you configure the ike/config file to require preshared keys. You enter keying material for ISAKMP SAs, that is, for IKE authentication, in the ike.preshared file. Because the preshared keys are used to authenticate the Phase 1 exchange, the file must be valid before the in.iked daemon starts.
- The ipseckeys file contains keying material for IPsec SAs. See "How to Manually Create IPsec Security Associations" on page 43 for examples of manually managing the file. The IKE daemon does not use this file. The keying material that IKE generates for IPsec SAs is stored in the kernel.

Note – Preshared keys cannot take advantage of hardware storage. Preshared keys are generated and stored on the system.

IKE Public Key Databases and Commands

The ikecert command manipulates the local system's publickey databases. You use this command when the ike/config file requires public key certificates. Because IKE uses these databases to authenticate the Phase 1 exchange, the databases must be populated before activating the in.iked daemon. Three subcommands handle each of the three databases: certlocal, certdb, and certrldb.

The ikecert command also handles key storage on the Sun Crypto Accelerator 4000 board. The tokens argument to the ikecert command lists the token IDs that are available on the board. The command finds the board through the PKCS #11 library that is specified in the /etc/inet/ike/config file. The PKCS #11 entry must be present. Otherwise, the -T option to the ikecert commands cannot work. The entry appears similar to the following:

```
pkcs11_path "/opt/SUNWconn/lib/libpkcs11.so"
```

For more information, see the ikecert(1M) man page.

`ikecert tokens` Command

The `tokens` argument lists the token IDs that are available on the Sun Crypto Accelerator 4000 board. Token IDs enable the `ikecert certlocal` and `ikecert certdb` commands to generate public key certificates and certificate requests on the board.

`ikecert certlocal` Command

The `certlocal` subcommand manages the private-key database. Options to this subcommand enable you to add, view, and remove private keys. This subcommand also creates either a self-signed certificate or a certificate request. The `-ks` option creates a self-signed certificate. The `-kc` option creates a certificate request. Keys are stored on the system in the `/etc/inet/secret/ike.privatekeys` directory, or on attached hardware with the `-T` option..

When you create a private key, the `ikecert` command relies on entries in the `ike/config` file. The correspondences between `ikecert` options and `ike/config` entries are shown in the following table.

TABLE 3–2 Correspondences Between `ikecert` Options and `ike/config` Entries

`ikecert` Options	**`ike/config` Entry**	**Notes**
`-A` *subject-alternate-name*	`cert_trust` *subject-alternate-name*	A nickname that uniquely identifies the certificate. Possible values are an IP address, an email address, or a domain name.
`-D` *X.509-distinguished-name*	*X.509-distinguished-name*	The full name of the certificate authority that includes the Country, Organization name, Organizational Unit, and Common Name.
`-t dsa-sha1`	`auth_method dss_sig`	An authentication method that is slightly slower than RSA. Is not patented.
`-t rsa-md5`, and `-t rsa-sha1`	`auth_method rsa_sig`	An authentication method that is slightly faster than DSA. Patent expired in September 2000. The RSA public key must be large enough to encrypt the biggest payload. Typically, an identity payload, such as the X.509 distinguished name, is the biggest payload.
`-t rsa-md5`, and `-t rsa-sha1`	`auth_method rsa_encrypt`	RSA encryption hides identities in IKE from eavesdroppers, but requires that the IKE peers know each other's public keys.

TABLE 3–2 Correspondences Between `ikecert` Options and `ike/config` Entries *(Continued)*

`ikecert` Options	`ike/config` Entry	Notes
`-T`	`pkcs11_path`	The PKCS #11 library handles key acceleration on the Sun Crypto Accelerator 1000 board and the Sun Crypto Accelerator 4000 board. The library also provides the tokens that handle key storage on the Sun Crypto Accelerator 4000 board.

If you issue a certificate request with the `ikecert certlocal -kc` command, you send the output of the command to a PKI organization or a certificate authority (CA). If your company runs its own PKI, you send the output to your PKI administrator. The PKI organization, the CA, or your PKI administrator then creates certificates. The certificates that the PKI or CA returns to you are input to the `certdb` subcommand. The CRL that the PKI returns to you is input for the `certrldb` subcommand.

`ikecert certdb` Command

The `certdb` subcommand manages the publickey database. Options to the subcommand enable you to add, view, and remove certificates and public keys. The command accepts, as input, certificates that were generated by the `ikecert certlocal -ks` command on a remote system. See "How to Configure IKE With Self-Signed Public Key Certificates" on page 71 for the procedure. This command also accepts the certificate that you receive from a PKI or CA as input. See "How to Configure IKE With Certificates Signed by a CA" on page 75 for the procedure.

On the system, the certificates and public keys are stored in the `/etc/inet/ike/publickeys` directory. The `-T` option stores the certificates, private keys, and public keys on attached hardware.

`ikecert certrldb` Command

The `certrldb` subcommand manages the certificate revocation list (CRL) database, `/etc/inet/ike/crls`. The `crls` database maintains the revocation lists for public keys. Certificates that are no longer valid are on this list. When PKIs provide you with CRLs, you install the CRLs in the CRL database with the `ikecert certrldb` command. See "How to Handle a Certificate Revocation List" on page 84 for the procedure.

/etc/inet/ike/publickeys Directory

The /etc/inet/ike/publickeys directory contains the public part of a public-private key pair and its certificate in files, or *slots*. The /etc/inet/ike directory is protected at 0755. The ikecert certdb command populates the directory. The -T option stores the keys on the Sun Crypto Accelerator 4000 board rather than in the publickeys directory.

The *slots* contain, in encoded form, the X.509 distinguished name of a certificate that was generated on another system. If you are using self-signed certificates, you use the certificate that you receive from the administrator of the remote system as input to the command. If you are using certificates from a PKI, you install two pieces of keying material from the PKI into this database. You install a certificate that is based on material that you sent to the PKI. You also install a CA from the PKI.

/etc/inet/secret/ike.privatekeys Directory

The /etc/inet/secret/ike.privatekeys directory holds private key files that are part of a public-private key pair, which is keying material for ISAKMP SAs. The directory is protected at 0700. The ikecert certlocal command populates the ike.privatekeys directory. Private keys are not effective until their public key counterparts, self-signed certificates or CAs, are installed. The public key counterparts are stored in the /etc/inet/ike/publickeys directory or on a Sun Crypto Accelerator 4000 board.

/etc/inet/ike/crls Directory

The /etc/inet/ike/crls directory contains certificate revocation list (CRL) files. Each file corresponds to a public certificate file in the /etc/inet/ike/publickeys directory. PKI organizations provide the CRLs for their certificates. You use the ikecert certrldb command to populate the database.

CHAPTER 4

Administering IKE (Tasks)

This chapter describes how to configure IKE for your systems. After IKE is configured, it automatically generates keying material for IPsec on your network. “Configuring IKE (Task Map)” on page 61 lists the tasks in this chapter.

For overview information about IKE, see Chapter 3. The `ikeadm`(1M), `ikecert`(1M), and `ike.config`(4) man pages contain useful procedures in their respective Examples sections.

Configuring IKE (Task Map)

Task	Description	For Instructions
Configure IKE with preshared keys	Protects communications between two systems by having them share a secret key.	“Configuring IKE With Preshared Keys (Task Map)” on page 62
Configure IKE with public key certificates	Protects communications with public key certificates. The certificates can be self-signed, or they can be vouched for by a PKI organization.	“Configuring IKE With Public Key Certificates (Task Map)” on page 71

Task	Description	For Instructions
Configure IKE to generate and store public key certificates on attached hardware	Enables a Sun Crypto Accelerator 1000 board or a Sun Crypto Accelerator 4000 board to accelerate IKE operations. Also enables the Sun Crypto Accelerator 4000 board to store public key certificates.	"Using Hardware With IKE (Task Map)" on page 86

For information on how to use roles to configure IKE, see "Role-Based Access Control (Tasks)" in *System Administration Guide: Security Services*.

Configuring IKE With Preshared Keys (Task Map)

Task	Description	For Instructions
Configure IKE with preshared keys	Involves creating a valid IKE policy file and `ike.preshared` file. IPsec files are also set up before booting the system to use the IKE-generated keys.	"How to Configure IKE With Preshared Keys" on page 63
Refresh preshared keys on a running IKE system	Involves checking the IKE privilege level and adding fresh keying material to the `ipseckeys` file on communicating systems.	"How to Refresh Existing Preshared Keys" on page 65
Add preshared keys to a running IKE system	Involves checking the IKE privilege level and running the `ikeadm` command with fresh keying material on communicating systems.	"How to Add a New Preshared Key" on page 66
Check that preshared keys are identical	Involves dumping the preshared keys on both systems.	"How to Verify That the Preshared Keys Are Identical" on page 70

▼ How to Configure IKE With Preshared Keys

The IKE implementation offers algorithms whose keys vary in length. The key length that you choose is determined by site security. In general, longer keys provide more security than shorter keys.

These procedures use the system names `enigma` and `partym`. Substitute the names of your systems for the names `enigma` and `partym`.

1. **On the system console, become superuser or assume an equivalent role.**

Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

---

2. **On each system, copy the file `/etc/inet/ike/config.sample` to the file `/etc/inet/ike/config`.**

3. **Enter rules and global parameters in the `ike/config` file on each system.**

   The rules and global parameters in this file should permit the IPsec policy in the system's `ipsecinit.conf` file to succeed. The following `ike/config` examples work with the `ipsecinit.conf` examples in "How to Secure Traffic Between Two Systems" on page 31.

   a. **For example, modify the `/etc/inet/ike/config` file on the `enigma` system:**

```
### ike/config file on enigma, 192.168.116.16

## Global parameters
#
## Phase 1 transform defaults
p1_lifetime_secs 14400
p1_nonce_len 40
#
## Defaults that individual rules can override.
p1_xform
  { auth_method preshared oakley_group 5 auth_alg sha encr_alg des }
p2_pfs 2
#
## The rule to communicate with partym

{ label "enigma-partym"       Label must be unique
  local_addr 192.168.116.16
  remote_addr 192.168.13.213
  p1_xform
   { auth_method preshared oakley_group 5 auth_alg md5 encr_alg 3des }
  p2_pfs 5
}
```

Note – All arguments to the `auth_method` parameter must be on the same line.

b. Modify the `/etc/inet/ike/config` file on the `partym` system:

```
### ike/config file on partym, 192.168.13.213
## Global Parameters
#
p1_lifetime_secs 14400
p1_nonce_len 40
#
p1_xform
  { auth_method preshared oakley_group 5 auth_alg sha encr_alg des }
p2_pfs 2

## The rule to communicate with enigma

{ label "partym-enigma"      Label must be unique
  local_addr 192.168.13.213
  remote_addr 192.168.116.16
  p1_xform
   { auth_method preshared oakley_group 5 auth_alg md5 encr_alg 3des }
  p2_pfs 5
}
```

4. On each system, check the validity of the file.

```
# /usr/lib/inet/in.iked -c -f /etc/inet/ike/config
```

5. Generate random numbers for use as keying material.

If your site has a random number generator, use that generator. On a Solaris system, you can use the `od` command. For example, the following command prints two lines of hexadecimal numbers:

```
% od -X -A n /dev/random | head -2
         f47cb0f4 32e14480 951095f8 2b735ba8
         0a9467d0 8f92c880 68b6a40e 0efe067d
```

For an explanation of the `od` command, see "How to Generate Random Numbers" on page 42 and the `od`(1) man page.

6. From the output of Step 5, construct one key.

```
f47cb0f432e14480951095f82b735ba80a9467d08f92c88068b6a40e
```

The authentication algorithm in this procedure is MD5, as shown in Step 3. The size of the hash, that is, the size of the authentication algorithm's output, determines the minimum recommended size of a preshared key. The output of the MD5 algorithm is 128 bits, or 32 characters. The example key is 56 characters long, which is longer than the recommended minimum.

7. Create the file `/etc/inet/secret/ike.preshared` on each system. Put the preshared key in each file.

a. **For example, on the `enigma` system, the `ike.preshared` file would appear similar to the following:**

```
# ike.preshared on enigma, 192.168.116.16
#...
{ localidtype IP
    localid 192.168.116.16
    remoteidtype IP
    remoteid 192.168.13.213
    # enigma and partym's shared key in hex (192 bits)
    key f47cb0f432e14480951095f82b735ba80a9467d08f92c88068b6a40e
    }
```

b. **On the `partym` system, the `ike.preshared` file would appear similar to the following:**

```
# ike.preshared on partym, 192.168.13.213
#...
{ localidtype IP
    localid 192.168.13.213
    remoteidtype IP
    remoteid 192.168.116.16
    # partym and enigma's shared key in hex (192 bits)
    key f47cb0f432e14480951095f82b735ba80a9467d08f92c88068b6a40e
    }
```

Note – The preshared keys on each system must be identical.

▼ How to Refresh Existing Preshared Keys

This procedure assumes that you want to replace an existing preshared key at regular intervals without rebooting. If you use a strong encryption algorithm, such 3DES or Blowfish, you might want to schedule key replacement for when you reboot both machines.

1. **On the system console, become superuser or assume an equivalent role.**

Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

---

2. **Generate random numbers and construct a key of the appropriate length.**

   For details, see "How to Generate Random Numbers" on page 42.

3. **Edit the `/etc/inet/secret/ike.preshared` file on each system, and replace the current key with a new key.**

For example, on the hosts `enigma` and `partym`, you would replace the value of `key` with a new number of the same length.

4. **Check that the `in.iked` daemon permits you to change keying material.**

   ```
   # /usr/sbin/ikeadm get priv
   Current privilege level is 0x2, access to keying material enabled
   ```

   You can change keying material if the command returns a privilege level of `0x1` or `0x2`. Level `0x0` does not permit keying material operations. By default, the `in.iked` daemon runs at the `0x0` level of privilege.

5. **If the `in.iked` daemon permits you to change keying material, read in the new version of the `ike.preshared` file.**

   ```
   # ikeadm read preshared
   ```

6. **If the `in.iked` daemon does not permit you to change keying material, kill the daemon and then restart the daemon.**

   ```
   # pkill in.iked
   # /usr/lib/inet/in.iked
   ```

   When the daemon restarts, the daemon reads the new version of the `ike.preshared` file.

▼ How to Add a New Preshared Key

You must have one preshared key for every policy entry in the `ipsecinit.conf` file. If you add a new policy entry while IPsec and IKE are running, the `in.iked` daemon can read in new keys. This procedure assumes the following:

- The systems are named `enigma` and `ada`. Substitute your system names for these names.
- The `in.iked` daemon is running on both systems.
- The interface that you want to protect with IPsec is included as an entry in the `/etc/hosts` file on both systems. The following entry is an example.

  ```
  192.168.15.7 ada
  ```

  This procedure also works with an IPv6 address in the `/etc/inet/ipnodes` file.
- You have added a new policy entry to the `/etc/inet/ipsecinit.conf` file on both systems. For example, the new entry on the `enigma` system appears similar to the following:

  ```
  {laddr enigma raddr ada} ipsec {auth_algs any encr_algs any sa shared}
  ```

  The entry on the `ada` system appears similar to the following:

  ```
  {laddr ada raddr enigma} ipsec {auth_algs any encr_algs any sa shared}
  ```

- You have created a rule for the `enigma` and `ada` systems to communicate securely in the `/etc/inet/ike/config` file on both systems. For example, the rule on the `enigma` system appears similar to the following:

```
### ike/config file on enigma, 192.168.116.16
...
## The rule to communicate with ada

{ label "enigma-to-ada"
  local_addr 192.168.116.16
  remote_addr 192.168.15.7
  p1_xform
  { auth_method preshared oakley_group 5 auth_alg md5 encr_alg blowfish }
  p2_pfs 5
    }
```

  The rule on the `ada` system appears similar to the following:

```
### ike/config file on ada, 192.168.15.7
...
## The rule to communicate with enigma

{ label "ada-to-enigma"
  local_addr 192.168.15.7
  remote_addr 192.168.116.16
  p1_xform
  { auth_method preshared oakley_group 5 auth_alg md5 encr_alg blowfish }
  p2_pfs 5
}
```

1. **On the system console, become superuser or assume an equivalent role.**

Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

---

2. **Check that the `in.iked` daemon permits you to change keying material.**

```
# /usr/sbin/ikeadm get priv
Current privilege level is 0x0, base privileges enabled
```

   You can change keying material if the command returns a privilege level of `0x1` or `0x2`. Level `0x0` does not permit keying material operations. By default, the `in.iked` daemon runs at the `0x0` level of privilege.

3. **If the `in.iked` daemon does not permit you to change keying material, kill the daemon. Then, restart the daemon with the correct privilege level.**

```
# pkill in.iked
# /usr/lib/inet/in.iked -p 2
Setting privilege level to 2!
```

4. **Generate random numbers and construct a key of 64 to 448 bits.**
   For details, see "How to Generate Random Numbers" on page 42.

5. **By some means, send the key to the administrator of the remote system.**
   You both need to add the same preshared key at the same time.

6. **Add the new keying material with the `add preshared` subcommand in `ikeadm` command mode.**

   ```
   ikeadm> add preshared { localidtype id-type localid id
   remoteidtype id-type remoteid id ike_mode mode key key }
   ```

   id-type Specifies the type of the *id*.

   id Specifies the IP address when *id-type* is IP.

   mode Specifies the IKE mode. The only accepted value is `main`.

   key Specifies the preshared key in hexadecimal format.

   a. **For example, on host `enigma`, you would add the key for the new interface, `ada`.**

      ```
      # ikeadm
      ikeadm> add preshared { localidtype ip localid 192.168.116.16
      remoteidtype ip remoteid 192.168.15.7 ike_mode main
      key 8d1fb4ee500e2bea071deb2e781cb48374411af5a9671714672bb1749ad9364d }
      ikeadm: Successfully created new preshared key.
      ```

   b. **On host `ada`, you would add the identical key.**

      ```
      # ikeadm
      ikeadm> add preshared { localidtype ip localid 192.168.15.7
      remoteidtype ip remoteid 192.168.116.16 ike_mode main
      key 8d1fb4ee500e2bea071deb2e781cb48374411af5a9671714672bb1749ad9364d }
      ikeadm: Successfully created new preshared key.
      ```

7. **Exit the `ikeadm` command mode.**

   ```
   ikeadm> exit
   #
   ```

8. **On each system, lower the privilege level of the `in.iked` daemon.**

   ```
   # ikeadm set priv base
   ```

9. **On each system, activate the `ipsecinit.conf` file to secure the added interface.**

   ```
   # ipsecconf -a /etc/inet/ipsecinit.conf
   ```

Caution – Read the warning when you execute the `ipsecconf` command. The same warning applies to restarting the `in.iked` daemon. A socket that is already latched, that is, the socket is in use, provides an unsecured back door into the system. For more extensive discussion, see "Security Considerations for `ipsecinit.conf` and `ipsecconf`" on page 24.

10. On each system, read in the new rules by using the `ikeadm` command.

```
# ikeadm read rules
```

A sample of the new rules for the `ada` and `enigma` systems is available at the start of the procedure. Because the rules are in the `/etc/inet/ike/config` file, the name of the file does not have to be specified to the `ikeadm` command.

11. To ensure that IKE preshared keys are available at reboot, add the keys to the `/etc/inet/secret/ike.preshared` file.

a. For example, on the `enigma` system, you would add the following keying information to the `ike.preshared` file:

```
# ike.preshared on enigma for the ada interface
#...
{ localidtype IP
  localid 192.168.116.16
  remoteidtype IP
  remoteid 192.168.15.7
  # enigma and ada's shared key in hex (32 - 448 bits required)
  key 8d1fb4ee500e2bea071deb2e781cb48374411af5a9671714672bb1749ad9364d
}
```

b. On the `ada` system, you would add the following keying information to the `ike.preshared` file:

```
# ike.preshared on ada for the enigma interface
#...
{ localidtype IP
  localid 192.168.15.7
  remoteidtype IP
  remoteid 192.168.116.16
  # ada and enigma's shared key in hex (32 - 448 bits required)
  key 8d1fb4ee500e2bea071deb2e781cb48374411af5a9671714672bb1749ad9364d
}
```

12. Verify that the systems can communicate. See "How to Verify That the Preshared Keys Are Identical" on page 70.

▼ How to Verify That the Preshared Keys Are Identical

If the preshared keys on the communicating systems are not identical, you see the following error message:

```
# rup system2
system2: RPC: Rpcbind failure
```

1. **On the system console, become superuser or assume an equivalent role.**

   Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

   ---

2. **Check that the `in.iked` daemon permits you to change keying material.**

   ```
   # /usr/sbin/ikeadm get priv
   Current privilege level is 0x0, base privileges enabled
   ```

   You can view keying material if the command returns a privilege level of `0x2`. Level `0x0` does not permit keying material operations. By default, the `in.iked` daemon runs at the `0x0` level of privilege.

3. **If the `in.iked` daemon does not permit you to view keying material, kill the daemon. Then, restart the daemon with the correct privilege level.**

   ```
   # pkill in.iked
   # /usr/lib/inet/in.iked -p 2
   Setting privilege level to 2!
   ```

4. **On each system, view the preshared key information.**

   ```
   # ikeadm dump preshared
   PSKEY: Preshared key (24 bytes): f47cb.../192
   LOCIP: AF_INET: port 0, 192.168.116.16 (enigma).
   REMIP: AF_INET: port 0, 192.168.13.213 (partym).
   ```

5. **Compare the two dumps.**

   If the preshared keys are not identical, replace one key with the other key in the `/etc/inet/secret/ike.preshared` file.

6. **When the verification is complete, lower the privilege level of the `in.iked` daemon.**

   ```
   # ikeadm set priv base
   ```

Configuring IKE With Public Key Certificates (Task Map)

Task	Description	For Instructions
Configure IKE with self-signed public key certificates	Involves creating self–signed certificates with the `ikecert certlocal -ks` command, and adding the public key from a remote system with the `ikecert certdb` command.	"How to Configure IKE With Self-Signed Public Key Certificates" on page 71
Configure IKE with a PKI Certificate Authority	Involves sending output from the `ikecert certlocal -kc` command to a PKI organization, and adding the public key, CA, and CRL from the organization.	"How to Configure IKE With Certificates Signed by a CA" on page 75
Configure public key certificates on local hardware	Involves one of: ■ Generating a self-signed certificate on the local hardware, and adding the public key from a remote system to the hardware. ■ Generating a certificate request on the local hardware, and adding the public key certificates from the PKI organization to the hardware.	"How to Generate and Store Public Key Certificates on Hardware" on page 80
Update the certificate revocation list (CRL) from the PKI organization	Involves accessing the CRL from a central distribution point.	"How to Handle a Certificate Revocation List" on page 84

▼ How to Configure IKE With Self-Signed Public Key Certificates

1. On the system console, become superuser or assume an equivalent role.

Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

2. Add a self-signed certificate to the `ike.privatekeys` database.

```
# ikecert certlocal -ks|-kc -m keysize -t keytype \
-D dname -A altname
```

`-ks` Creates a self-signed certificate.

`-kc` Creates a certificate request. For the procedure, see "How to Configure IKE With Certificates Signed by a CA" on page 75.

keysize Is the size of the key. The *keysize* can be 512, 1024, 2048, 3072, or 4096.

keytype Specifies the type of algorithm to use. The *keytype* can be `rsa-sha1`, `rsa-md5`, or `dsa-sha1`.

dname Is the X.509 distinguished name for the certificate subject. The *dname* typically has the form: C=country, O=organization, OU=organizational unit, CN=common name. Valid tags are C, O, OU, and CN.

altname Is the alternate name for the certificate. The *altname* is in the form of `tag=value`. Valid tags are `IP`, `DNS`, `EMAIL`, `URI`, `DN`, and `RID`.

a. For example, the command on the `partym` system would appear similar to the following:

```
# ikecert certlocal -ks -m 1024 -t rsa-md5 \
> -D "C=US, O=PartyCompany, OU=US-Partym, CN=Partym" \
> -A IP=192.168.13.213
Creating software private keys.
  Writing private key to file /etc/inet/secret/ike.privatekeys/0.
Enabling external key providers - done.
Acquiring private keys for signing - done.
Certificate:
 Proceeding with the signing operation.
 Certificate generated successfully (.../publickeys/0)
Finished successfully.
Certificate added to database.
-----BEGIN X509 CERTIFICATE-----
MIICLTCCAZagAwIBAgIBATANBgkqhkiG9w0BAQQFADBNMQswCQYDVQQGEwJVUzEX
...
6sKTxpg4GP3GkQGcd0r1rhW/3yaWBkDwOdFCqEUyffzU
-----END X509 CERTIFICATE-----
```

b. The command on the `enigma` system would appear similar to the following:

```
# ikecert certlocal -ks -m 1024 -t rsa-md5 \
> -D "C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax" \
> -A IP=192.168.116.16
```

```
Creating software private keys.
  ...
Certificate added to database.
-----BEGIN X509 CERTIFICATE-----
MIICKDCCAZGgAwIBAgIBATANBgkqhkiG9w0BAQQFADBJMQswCQYDVQQGEwJVUzEV
...
jpxfLM98xyFVyLCbkr3dZ3Tvxvi732BXePKF2A==
-----END X509 CERTIFICATE-----
```

3. **Save the certificate, and send it to the remote system.**

   You can paste the certificate into an email.

   a. **For example, you would send the following `partym` certificate to the `enigma` administrator:**

   ```
   To: admin@ja.enigmaexample.com
   From: admin@us.partyexample.com
   Message: -----BEGIN X509 CERTIFICATE-----
   MIICLTCCAZagAwIBAgIBATANBgkqhkiG9w0BAQQFADBNMQswCQYDVQQGEwJVUzEX
   ...
   6sKTxpg4GP3GkQGcd0r1rhW/3yaWBkDwOdFCqEUyffzU
   -----END X509 CERTIFICATE-----
   ```

   b. **The `enigma` administrator would send you the following `enigma` certificate:**

   ```
   To: admin@us.partyexample.com
   From: admin@ja.enigmaexample.com
   Message: -----BEGIN X509 CERTIFICATE-----
   MIICKDCCAZGgAwIBAgIBATANBgkqhkiG9w0BAQQFADBJMQswCQYDVQQGEwJVUzEV
   ...
   jpxfLM98xyFVyLCbkr3dZ3Tvxvi732BXePKF2A==
   -----END X509 CERTIFICATE-----
   ```

4. **On each system, edit the `/etc/inet/ike/config` file to recognize the certificates.**

   The administrator of the remote system provides the values for the `cert_trust`, `remote_addr`, and `remote_id` parameters.

   a. **For example, on the `partym` system, the `ike/config` file would appear similar to the following:**

   ```
   # Explicitly trust the following self-signed certs
   # Use the Subject Alternate Name to identify the cert

   cert_trust "192.168.13.213"
   cert_trust "192.168.116.16"

   ## Parameters that may also show up in rules.

   p1_xform
     { auth_method preshared oakley_group 5 auth_alg sha encr_alg des }
   p2_pfs 5
   ```

```
{
 label "US-partym to JA-enigmax"
 local_id_type dn
 local_id "C=US, O=PartyCompany, OU=US-Partym, CN=Partym"
 remote_id "C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax"

 local_addr  192.168.13.213
 remote_addr 192.168.116.16

 p1_xform
  {auth_method rsa_encrypt oakley_group 2 auth_alg md5 encr_alg 3des}
}
```

b. On the `enigma` system, add `enigma` values for local parameters in the `ike/config` file.

For the remote parameters, use `partym` values. Ensure that the value for the `label` keyword is unique. The value must be different from the remote system's `label` value.

```
...
{
 label "JA-enigmax to US-partym"
 local_id_type dn
 local_id "C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax"
 remote_id "C=US, O=PartyCompany, OU=US-Partym, CN=Partym"

 local_addr  192.168.116.16
 remote_addr 192.168.13.213
...
```

5. On each system, add the certificate that you received.

a. Copy the public key from the administrator's email.

b. Type the `ikecert certdb -a` command and press the Return key.

No prompts display when you press the Return key.

```
# ikecert certdb -a
     Press the Return key
```

c. Paste the public key. Then press the Return key. To end the entry, press Control-D.

```
-----BEGIN X509 CERTIFICATE-----
MIIC...
...
----END X509 CERTIFICATE-----
     Press the Return key
<Control>-D
```

6. Verify with the other administrator that the keys have not been tampered with.

For example, you can phone the other administrator to compare the values of the public key hash. The public key hash for the shared certificate should be identical on the two systems.

a. For example, on the `partym` system, list the stored certificates.

```
partym # ikecert certdb -l
Certificate Slot Name: 0   Type: rsa-md5
    Subject Name: <C=US, O=PartyCompany, OU=US-Partym, CN=Partym>
    Key Size: 1024
    Public key hash: B2BD13FCE95FD27ECE6D2DCD0DE760E2

Certificate Slot Name: 1   Type: rsa-md5
    (Private key in certlocal slot 0)
    Subject Name: <C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax>
    Key Size: 1024
    Public key hash: 2239A6A127F88EE0CB40F7C24A65B818
```

b. On the `enigma` system, list the stored certificates.

```
enigma # ikecert certdb -l
Certificate Slot Name: 4   Type: rsa-md5
    Subject Name: <C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax>
    Key Size: 1024
    Public key hash: DF3F108F6AC669C88C6BD026B0FCE3A0

Certificate Slot Name: 5   Type: rsa-md5
    (Private key in certlocal slot 4)
    Subject Name: <C=US, O=PartyCompany, OU=US-Partym, CN=Partym>
    Key Size: 1024
    Public key hash: 2239A6A127F88EE0CB40F7C24A65B818
```

Note – In this example, the public key hash is different from the public key hash that your systems generate.

▼ How to Configure IKE With Certificates Signed by a CA

1. On the system console, become superuser or assume an equivalent role.

Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

---

2. Use the `ikecert certlocal -kc` command to create a certificate request.

For a description of the arguments to the command, see Step 2 in "How to Configure IKE With Self-Signed Public Key Certificates" on page 71.

```
# ikecert certlocal -kc -m keysize -t keytype \
-D dname -A altname
```

a. For example, the following command creates a certificate request on the `partym` system:

```
# ikecert certlocal -kc -m 1024 -t rsa-md5 \
> -D "C=US, O=PartyCompany\, Inc., OU=US-Partym, CN=Partym" \
> -A "DN=C=US, O=PartyCompany\, Inc., OU=US-Partym"
Creating software private keys.
  Writing private key to file /etc/inet/secret/ike.privatekeys/2.
Enabling external key providers - done.
Certificate Request:
  Proceeding with the signing operation.
  Certificate request generated successfully (.../publickeys/0)
Finished successfully.
-----BEGIN CERTIFICATE REQUEST-----
MIIByjCCATMCAQAwUzELMAkGA1UEBhMCVVMxHTAbBgNVBAoTFEV4YW1wbGVDb21w
...
lcM+tw0ThRrfuJX9t/Qa1R/KxRlMA3zckO80mO9X
-----END CERTIFICATE REQUEST-----
```

b. The following command creates a certificate request on the `enigma` system:

```
# ikecert certlocal -kc -m 1024 -t rsa-md5 \
> -D "C=JA, O=EnigmaCo\, Inc., OU=JA-Enigmax, CN=Enigmax" \
> -A "DN=C=JA, O=EnigmaCo\, Inc., OU=JA-Enigmax"
Creating software private keys.
...
Finished successfully.
-----BEGIN CERTIFICATE REQUEST-----
MIIBuDCCASECAQAwSTELMAkGA1UEBhMCVVMxFTATBgNVBAoTDFBhcnR5Q29tcGFu
...
8qlqdjaStLGfhDOO
-----END CERTIFICATE REQUEST-----
```

3. Submit the certificate request to a PKI organization.

The PKI organization can tell you how to submit the certificate request. Most organizations have a web site with a submission form. The form requires proof that the submission is legitimate. Typically, you paste your certificate request into the form. When your request has been checked by the organization, the organization issues you the following two certificate objects and a list of revoked certificates:

- Your public key certificate – This certificate is based on the request that you submitted to the organization. The request that you submitted is part of this public key certificate. The certificate uniquely identifies you.
- A Certificate Authority – The organization's signature. The CA verifies that your public key certificate is legitimate.

- A Certificate Revocation List (CRL) – The latest list of certificates that the organization has revoked. The CRL is not sent separately as a certificate object if access to the CRL is embedded in the public key certificate.

  When a URI for the CRL is embedded in the public key certificate, IKE can automatically retrieve the CRL for you. Similarly, when a DN (directory name on an LDAP server) entry is embedded in the public key certificate, IKE can retrieve and cache the CRL from an LDAP server that you specify.

  See "How to Handle a Certificate Revocation List" on page 84 for an example of an embedded URI and an embedded DN entry in a public key certificate.

4. **Add each certificate to your system by using the `ikecert certdb -a` command.**

   The `-a` option adds the pasted object to the appropriate certificate database on your system. For more information, see "IKE With Public Key Certificates" on page 51.

   a. **On the system console, become superuser or assume an equivalent role.**

   b. **Add the public key certificate that you received from the PKI organization.**

   ```
   # ikecert certdb -a
       Press the Return key
       Paste the certificate:
   -----BEGIN X509 CERTIFICATE-----
   ...
   -----END X509 CERTIFICATE----
       Press the Return key
   <Control>-D
   ```

   c. **Add the CA from the PKI organization.**

   ```
   # ikecert certdb -a
       Press the Return key
       Paste the CA:
   -----BEGIN X509 CERTIFICATE-----
   ...
   -----END X509 CERTIFICATE----
       Press the Return key
   <Control>-D
   ```

   d. **If the PKI organization has sent a list of revoked certificates, add the CRL to the `certrldb` database:**

   ```
   # ikecert certrldb -a
       Press the Return key
       Paste the CRL:
   -----BEGIN CRL-----
   ...
   -----END CRL----
       Press the Return key
   <Control>-D
   ```

5. **Edit the `/etc/inet/ike/config` file to recognize the PKI organization.**

   Use the name that the PKI organization provides.

   a. **For example, the `ike/config` file on the `partym` system might appear similar to the following:**

```
# Trusted root cert
# This certificate is from Example PKI
# This is the X.509 distinguished name for the CA that it issues.

cert_root "C=US, O=ExamplePKI\, Inc., OU=PKI-Example, CN=Example PKI"

## Parameters that may also show up in rules.

p1_xform
 { auth_method rsa_sig oakley_group 1 auth_alg sha1 encr_alg des }
p2_pfs 2

{
 label "US-partym to JA-enigmax - Example PKI"
 local_id_type dn
 local_id  "C=US, O=PartyCompany, OU=US-Partym, CN=Partym"
 remote_id "C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax"

 local_addr  192.168.13.213
 remote_addr 192.168.116.16

 p1_xform
  {auth_method rsa_encrypt oakley_group 2 auth_alg md5 encr_alg 3des}
}
```

   Note – All arguments to the `auth_method` parameter must be on the same line.

   b. **On the `enigma` system, use `enigma` values for local parameters and `partym` values for remote parameters.**

      Ensure that the value for the `label` keyword is unique. The value must be different from the remote system's `label` value.

```
...
{
 label "JA-enigmax to US-partym - Example PKI"
 local_id_type dn
 local_id   "C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax"
 remote_id  "C=US, O=PartyCompany, OU=US-Partym, CN=Partym"

 local_addr  192.168.116.16
 remote_addr 192.168.13.213
...
```

6. **If the PKI organization does not provide a CRL, add the keyword `ignore_crls` to the `ike/config` file.**

```
# Trusted root cert
...
cert_root "C=US, O=ExamplePKI\, Inc., OU=PKI-Example, CN=Example PKI"
ignore_crls
...
```

The `ignore_crls` keyword tells IKE not to search for CRLs.

7. **If the PKI organization provides a central distribution point for CRLs, you can modify the `ike/config` file to point to that location.**

See "How to Handle a Certificate Revocation List" on page 84 for examples.

Note – The following steps are necessary only if `rsa_encrypt` is the `auth_method` in the `/etc/inet/ike/config` file.

---

8. **Because the `auth_method` parameter is set to `rsa_encrypt`, add the peer's certificate to the `publickeys` database.**

   a. **Send the certificate to the remote system's administrator.**

   You can paste the certificate into an email.

      i. **For example, the `partym` administrator would send the following email:**

```
To: admin@ja.enigmaexample.com
From: admin@us.partyexample.com
Message: -----BEGIN X509 CERTIFICATE-----
MII...
----END X509 CERTIFICATE-----
```

      ii. **The `enigma` administrator would send the following email:**

```
To: admin@us.partyexample.com
From: admin@ja.enigmaexample.com
Message: -----BEGIN X509 CERTIFICATE-----
MII
...
-----END X509 CERTIFICATE-----
```

   b. **On each system, add the emailed certificate to the local `publickeys` database.**

```
# ikecert certdb -a
     Press the Return key
-----BEGIN X509 CERTIFICATE-----
MII...
-----END X509 CERTIFICATE-----
     Press the Return key
```

```
<Control>-D
```

The authentication method for RSA encryption hides identities in IKE from eavesdroppers. Because the `rsa_encrypt` method hides identities, IKE does not know the peer. Therefore, IKE cannot retrieve the peer's certificate. As a result, this method requires that the IKE peers know each other's public keys. Therefore, when you use an `auth_method` of `rsa_encrypt` in the `/etc/inet/ike/config` file, you must add the peer's certificate to the `publickeys` database. So, the `publickeys` database then holds three certificates for each communicating pair of systems:

- Your public key certificate
- The CA certificate
- The peer's public key certificate

▼ How to Generate and Store Public Key Certificates on Hardware

Prerequisites for generating and storing public keys and public key certificates on hardware include the following:

- The hardware must be configured.
- The `/etc/inet/ike/config` file must point to a library that is implemented according to the following standard: RSA Security Inc. PKCS #11 Cryptographic Token Interface (Cryptoki), that is, a PKCS #11 library.

  See "How to Use the Sun Crypto Accelerator 4000 Board With IKE" on page 87 for setup instructions.

Generating and storing public key certificates on hardware is similar to generating and storing public key certificates on your system. There are two differences:

- The `ikecert certlocal` and `ikecert certdb` commands must identify the hardware. The `-T` option with the token ID identifies the hardware to the commands.
- The `/etc/inet/ike/config` file must point to the hardware with the `pkcs11_path` keyword.

1. **On the system console, become superuser or assume an equivalent role.**

Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

---

2. **Generate a self-signed certificate or a certificate request, and specify the token ID. Choose one of the following options:**

Note – The Sun Crypto Accelerator 4000 board supports keys up to 2048 bits for RSA. For DSA, this board supports keys up to 1024 bits.

- For a self-signed certificate, use this syntax.

```
# ikecert certlocal -ks -m 1024 -t rsa-md5 \
> -D "C=US, O=PartyCompany, OU=US-Partym, CN=Partym" \
> -a -T SUN-4000-stor IP=192.168.116.16
Creating hardware private keys.
Enter PIN for PKCS#11 token:     Type user:password
-----BEGIN X509 CERTIFICATE-----
MIIBwjCCASsCBD9bz5swDQYJKoZIhvcNAQEEBQAwKDELMAkGA1UEBhMCVVMxGTAX
...
PiktCuvURc1TXswaFyftzmLKWafUOQ==
-----END X509 CERTIFICATE-----
```

  The argument to the `-T` option is the token ID from the attached Sun Crypto Accelerator 4000 board.

- For a certificate request, use this syntax.

```
# ikecert certlocal -kc -m 1024 -t rsa-md5 \
> -D "C=US, O=PartyCompany, OU=US-Partym, CN=Partym" \
> -a -T SUN-4000-stor IP=192.168.116.16
Creating hardware private keys.
Enter PIN for PKCS#11 token:     Type user:password
-----BEGIN X509 CERTIFICATE-----
MIIBuDCCASECAQAwSTELMAkGA1UEBhMCVVMxFTATBgNVBAoTDFBhcnR5Q29tcGFu
...
oKUDBbZ9O/pLWYGr
-----END X509 CERTIFICATE-----
```

For a description of the arguments to the `ikecert` command, see the `ikecert`(1M) man page

3. **At the prompt for a PIN, type the Sun Crypto Accelerator 4000 user, a colon, and the user's password.**

   If the Sun Crypto Accelerator 4000 board has a user `ikemgr` whose password is `rgm4tigt`, you would type the following:

```
Enter PIN for PKCS#11 token: ikemgr:rgm4tigt
```

Note – The PIN response is stored on disk *as clear text*.

---

4. **Send your certificate for use by the other party. Choose one of the following options:**

- Send the self-signed certificate to the remote system. You can paste the certificate into an email.
- Send the certificate request to an organization that handles PKI. Follow the instructions of the PKI organization to submit the certificate request. For a more detailed discussion, see Step 3 of "How to Configure IKE With Certificates Signed by a CA" on page 75.

5. **On your system, edit the `/etc/inet/ike/config` file to recognize the certificates. Choose one of the following options.**

   - For a self-signed certificate, use the values that the administrator of the remote system provides for the `cert_trust`, `remote_id`, and `remote_addr` parameters.

     For example, on the `enigma` system, the `ike/config` file would appear similar to the following:

```
# Explicitly trust the following self-signed certs
# Use the Subject Alternate Name to identify the cert

cert_trust "192.168.116.16"       Local system's certificate
cert_trust "192.168.13.213"       Remote system's certificate

pkcs11_path "/opt/SUNWconn/lib/libpkcs11.so"      Hardware connection
...
{
 label "JA-enigmax to US-partym"
 local_id_type dn
 local_id "C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax"
 remote_id "C=US, O=PartyCompany, OU=US-Partym, CN=Partym"

 local_addr  192.168.116.16
 remote_addr 192.168.13.213

 p1_xform
  {auth_method rsa_encrypt oakley_group 2 auth_alg md5 encr_alg 3des}
}
```

   - For a certificate request, enter the name that the PKI organization provides as the value for the `cert_root` keyword.

     For example, the `ike/config` file on the `enigma` system might appear similar to the following:

```
# Trusted root cert
# This certificate is from Example PKI
# This is the X.509 distinguished name for the CA that it issues.

cert_root "C=US, O=ExamplePKI\, Inc., OU=PKI-Example, CN=Example PKI"

pkcs11_path "/opt/SUNWconn/lib/libpkcs11.so"      Hardware connection
...
{
 label "JA-enigmax to US-partym - Example PKI"
```

```
 local_id_type dn
 local_id "C=JA, O=EnigmaCo, OU=JA-Enigmax, CN=Enigmax"
 remote_id  "C=US, O=PartyCompany, OU=US-Partym, CN=Partym"

 local_addr  192.168.116.16
 remote_addr 192.168.13.213

 p1_xform
  {auth_method rsa_encrypt oakley_group 2 auth_alg md5 encr_alg 3des}
}
```

6. **Place the certificates from the other party in the hardware.**

   Respond to the PIN request as you responded in Step 3.

Note – You *must* add the public key certificates to the same attached hardware that generated your private key.

- For a self-signed certificate, add the remote system's self-signed certificate.

  ```
  # ikecert certdb -a -T SUN-4000-stor
      Press the Return key
      Paste the self-signed certificate
  <Control>-D
  Enter PIN for PKCS#11 token:      Type user:password
  ```

  If you used `rsa_encrypt` as the value for the `auth_method` parameter for a self-signed certificate, add the peer's certificate to the hardware store.

  ```
  # ikecert certdb -a -T SUN-4000-stor
      Press the Return key
      Paste the peer's certificate
  <Control>-D
  Enter PIN for PKCS#11 token:      Type user:password
  ```

- For certificates from a PKI organization, add the certificate that the organization generated from your certificate request, and add the certificate authority (CA).

  ```
  # ikecert certdb -a -T SUN-4000-stor
      Press the Return key
      Paste the returned certificate
  <Control>-D
  Enter PIN for PKCS#11 token:      Type user:password
  ```

  ```
  # ikecert certdb -a -T SUN-4000-stor
      Press the Return key
      Paste the CA certificate
  <Control>-D
  Enter PIN for PKCS#11 token:      Type user:password
  ```

  To add a certificate revocation list (CRL) from the PKI organization, see "How to Handle a Certificate Revocation List" on page 84.

▼ How to Handle a Certificate Revocation List

A certificate revocation list (CRL) contains outdated or compromised certificates from a Certificate Authority. You have four ways to handle CRLs.

- If your CA organization does not issue CRLs, you can instruct IKE to ignore CRLs in your `/etc/inet/ike/config` file. This option is shown in Step 6 in "How to Configure IKE With Certificates Signed by a CA" on page 75.
- IKE can access the CRLs from a URI (uniform resource indicator) whose address is embedded in the public key certificate from the CA.
- IKE can access the CRLs from an LDAP server whose DN (directory name) entry is embedded in the public key certificate from the CA.
- You can provide the CRL as an argument to the `ikecert certrldb` command.

The following procedure describes how to instruct IKE to use CRLs from a central distribution point.

1. **Display the certificate that you received from the CA.**

   `# ikecert certdb -lv` *certspec*

   `-l` Lists certificates in the IKE certificate database.

   `-v` Lists the certificates in verbose mode. Use this option with care.

   certspec Is a pattern that matches a certificate in the IKE certificate database.

   For example, the following certificate was issued by Sun Microsystems. Details have been altered.

```
# ikecert certdb -lv example-protect.sun.com
Certificate Slot Name: 0   Type: dsa-sha1
   (Private key in certlocal slot 0)
 Subject Name: <O=Sun Microsystems Inc, CN=example-protect.sun.com>
 Issuer Name: <CN=Sun Microsystems Inc CA (Cl B), O=Sun Microsystems Inc>
 SerialNumber: 14000D93
   Validity:
      Not Valid Before: 2002 Jul 19th, 21:11:11 GMT
      Not Valid After:  2005 Jul 18th, 21:11:11 GMT
   Public Key Info:
      Public Modulus  (n) (2048 bits): C575A...A5
      Public Exponent (e) (  24 bits): 010001
   Extensions:
      Subject Alternative Names:
              DNS = example-protect.sun.com
      Key Usage: DigitalSignature KeyEncipherment
      [CRITICAL]
   CRL Distribution Points:
      Full Name:
         URI = #Ihttp://www.sun.com/pki/pkismica.crl#i
         DN = <CN=Sun Microsystems Inc CA (Cl B), O=Sun Microsystems Inc>
      CRL Issuer:
      Authority Key ID:
```

```
Key ID:              4F ... 6B
SubjectKeyID:        A5 ... FD
Certificate Policies
Authority Information Access
```

Notice the `CRL Distribution Points` data. The URI entry indicates that this organization's CRL is available on the web. The DN entry indicates that the CRL is also available on an LDAP server. You can use one of these two options.

2. **To use the URI, add the keyword `use_http` to the host's `/etc/inet/ike/config` file.**

   For example, the `ike/config` file would appear similar to the following:

   ```
   # Use CRL from organization's URI
   use_http
   ...
   ```

   You can also use a web proxy by adding the keyword `proxy` in the `ike/config` file. The `proxy` keyword takes a URL as an argument, as in the following:

   ```
   proxy "http://proxy1:8080"
   ```

   IKE retrieves the CRL and caches the CRL until the certificate expires.

3. **To use LDAP, use the LDAP server as an argument to the `ldap-list` keyword in the host's `/etc/inet/ike/config` file.**

   Your organization provides the name of the LDAP server. The entry in the `ike/config` file would appear similar to the following:

   ```
   # Use CRL from organization's LDAP
   ldap-list "ldap1.sun.com:389,ldap2.sun.com"
   ...
   ```

   IKE retrieves the CRL and caches the CRL until the certificate expires.

Example—Pasting a CRL Into the Local `certrldb` Database

If the PKI organization's CRL is not available from a central distribution point, you can add the PKI organization's CRL manually to the local `certrldb` database. Follow the PKI organization's instructions for extracting the CRL, then add the CRL to the database with the `ikecert certrldb -a` command.

```
# ikecert certrldb -a
    Press the Return key
    Paste the CRL from the PKI organization
    Press the Return key
    Press <Control>-D to enter the CRL into the database
```

Using Hardware With IKE (Task Map)

Task	Description	For Instructions
Off-load IKE key operations to the Sun Crypto Accelerator 1000 board	Involves setting the path to the PKCS#11 library.	"How to Use the Sun Crypto Accelerator 1000 Board With IKE" on page 86
Off-load IKE key operations and store the keys on the Sun Crypto Accelerator 4000 board	Involves setting the path to the PKCS#11 library, and listing the available token IDs.	"How to Use the Sun Crypto Accelerator 4000 Board With IKE" on page 87

▼ How to Use the Sun Crypto Accelerator 1000 Board With IKE

Note – The following procedure assumes that a Sun Crypto Accelerator 1000 board is attached to the system. The procedure also assumes that the software for the board has been installed and that the software has been configured. For instructions, see the *Sun Crypto Accelerator 1000 Board Version 1.1 Installation and User's Guide*.

1. **On the system console, become superuser or assume an equivalent role.**

   Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

2. **Add the PKCS #11 library path to the `/etc/inet/ike/config` file.**

   ```
   pkcs11_path "/opt/SUNWconn/lib/libpkcs11.so"
   ```

   The path name must point to a 32-bit PKCS #11 library. If the library is present, IKE uses the library's routines to accelerate IKE public key operations on the Sun Crypto Accelerator 1000 board. When the board handles these expensive operations, operating system resources are free for other operations.

3. **Close the file and reboot.**

4. **After rebooting, check that the library has been linked. Type the following command to determine whether a PKCS #11 library has been linked:**

```
# ikeadm get stats
Phase 1 SA counts:
Current:   initiator:          0   responder:          0
Total:     initiator:          0   responder:          0
Attempted: initiator:          0   responder:          0
Failed:    initiator:          0   responder:          0
           initiator fails include 0 time-out(s)
PKCS#11 library linked in from /opt/SUNWconn/lib/libpkcs11.so
#
```

Unlike other parameters in the `/etc/inet/ike/config` file, the `pkcs11_path` keyword is read only when IKE is started. If you use the `ikeadm` command to add or reload a new `/etc/inet/ike/config` file, the `pkcs11_path` persists. The path persists because the IKE daemon does not clobber data from the Phase 1 exchange. Keys that are accelerated by PKCS #11 are part of Phase 1 data.

▼ How to Use the Sun Crypto Accelerator 4000 Board With IKE

Note – The following procedure assumes that a Sun Crypto Accelerator 4000 board is attached to the system. The procedure also assumes that the software for the board has been installed and that the software has been configured. For instructions, see the *Sun Crypto Accelerator 4000 Board Installation and User's Guide*. The guide is available from the Sun Hardware Documentation web site, under Network and Security Products.

---

1. **On the system console, become superuser or assume an equivalent role.**

   Note – Logging in remotely exposes security-critical traffic to eavesdropping. Even if you somehow protect the remote login, the security of the system is reduced to the security of the remote login session.

   ---

2. **Add the PKCS #11 library path to the `/etc/inet/ike/config` file.**

   ```
   pkcs11_path "/opt/SUNWconn/lib/libpkcs11.so"
   ```

   The path name must point to a 32-bit PKCS #11 library. If the library is present, IKE uses the library's routines to handle key generation and key storage on the Sun Crypto Accelerator 4000 board.

3. **Close the file and reboot.**

4. **After rebooting, check that the library has been linked. Type the following command to determine whether a PKCS #11 library has been linked:**

```
$ ikeadm get stats
...
PKCS#11 library linked in from /opt/SUNWconn/lib/libpkcs11.so
$
```

Unlike other parameters in the `/etc/inet/ike/config` file, the `pkcs11_path` keyword is read only when IKE is started. If you use the `ikeadm` command to add or reload a new `/etc/inet/ike/config` file, the `pkcs11_path` persists. The path persists because the IKE daemon does not clobber Phase 1 data.

Note – The Sun Crypto Accelerator 4000 board supports keys up to 2048 bits for RSA. For DSA, this board supports keys up to 1024 bits.

---

5. **Find the token ID for the attached Sun Crypto Accelerator 4000 board.**

```
$ ikecert tokens
Available tokens with library "/opt/SUNWconn/lib/libpkcs11.so":

"SUN-1000-accel                "
"SUN-4000-stor                 "
```

The library returns a token ID, also called a keystore name, of 32 characters. In this example, you could use the `SUN-4000-stor` token with the `ikecert` commands to store IKE keys

For instructions on how to use the token, see "How to Generate and Store Public Key Certificates on Hardware" on page 80.

The trailing spaces are automatically padded by the `ikecert` command.

APPENDIX **A**

IPsec and IKE Administration Guide Updates

The following sections list the updates to Solaris 9 operating environment features that are described in this book.

Solaris 9 4/03 Updates

IKE encryption can be accelerated with hardware. See "How to Use the Sun Crypto Accelerator 1000 Board With IKE" on page 86.

Solaris 9 12/03 Updates

- IKE runs over IPv6 networks as well as over IPv4 networks.
- IKE public keys, private keys, and certificates can be stored on a Sun Crypto Accelerator 4000 board. The board can also accelerate IKE encryption. See the following sections for more information:
  - "IKE and Hardware Storage" on page 53
  - "How to Generate and Store Public Key Certificates on Hardware" on page 80
  - "How to Use the Sun Crypto Accelerator 4000 Board With IKE" on page 87

Glossary

This glossary contains definitions of network security terms.

3DES See Triple-DES.

AES Advanced Encryption Standard. A symmetric 128-bit block data encryption technique. The U.S. government adopted the Rijndael variant of the algorithm as its encryption standard in October 2000. AES replaces DES encryption as the government standard.

asymmetric key cryptography An encryption system in which the sender and receiver of a message use different keys to encrypt and decrypt the message. Asymmetric keys are used to establish a secure channel for symmetric key encryption. Diffie-Hellman is an example of an asymmetric key protocol. Contrast with symmetric key cryptography.

authentication header An extension header that provides authentication and integrity, without confidentiality, to IP datagrams.

bidirectional tunnel A tunnel that can transmit datagrams in both directions.

Blowfish A symmetric block cipher algorithm that takes a variable-length key from 32 bits to 448 bits. Its author, Bruce Schneier, claims that Blowfish is optimized for applications where the key does not change often.

CA See certificate authority (CA).

certificate authority (CA) A trusted third-party organization or company that issues digital certificates used to create digital signatures and public-private key pairs. The CA guarantees the identity of the individual who is granted the unique certificate.

certificate revocation list (CRL) A list of public key certificates that have been revoked by a CA.

DES Data Encryption Standard. A symmetric-key encryption method developed in 1975 and standardized by ANSI in 1981 as ANSI X.3.92. DES uses a 56-bit key.

digital signature	A digital code that is attached to an electronically transmitted message that uniquely identifies the sender.
DSA	Digital Signature Algorithm. A public key algorithm with a variable key size from 512 to 4096 bits. The U.S. Government standard, DSS, goes up to 1024 bits. DSA relies on SHA-1 for input.
Diffie-Hellman protocol	Also known as public key cryptography. An asymmetric cryptographic key agreement protocol that was developed by Diffie and Hellman in 1976. The protocol enables two users to exchange a secret key over an insecure medium without any prior secrets. Diffie-Hellman is used by the IKE protocol.
encapsulating security payload (ESP)	An extension header that provides integrity and confidentiality to datagrams.
encapsulation	The process of a header and a payload being placed in the first packet, which is subsequently placed in the second packet's payload.
firewall	Any device or software that protects an organization's private network or intranet from intrusion by external networks such as the Internet.
hash value	A number that is generated from a string of text. Hash functions are used to ensure that transmitted messages have not been tampered with. MD5 and SHA-1 are examples of one-way hash functions.
HMAC	Keyed hashing method for message authentication. HMAC is a secret key authentication algorithm. HMAC is used with an iterative cryptographic hash function, such as MD5 or SHA-1, in combination with a secret shared key. The cryptographic strength of HMAC depends on the properties of the underlying hash function.
IKE	Internet Key Exchange. IKE automates the provision of authenticated keying material for IPsec security association (SA)s.
IP datagram	A packet of information that is carried over IP. An IP datagram contains a header and data. The header includes the addresses of the source and the destination of the datagram. Other fields in the header help identify and recombine the data with accompanying datagrams at the destination.
IP in IP encapsulation	The mechanism for tunneling IP packets within IP packets.
IPsec	The IP security architecture (IPsec) that provides protection for IP datagrams.
IPv4	Internet Protocol, version 4. IPv4 is sometimes referred to as IP. This version supports a 32-bit address space.
IPv6	Internet Protocol, version 6. This version supports a 128-bit address space.
key management	The way in which you manage security association (SA)s.

keystore name The name that an administrator gives to the storage area, or keystore, on a network interface card (NIC). The keystore name is also called the token or the token ID.

message authentication code (MAC) MAC provides assurance of data integrity and authenticates data origin. MAC does not protect against eavesdropping.

MD5 An iterative cryptographic hash function that is used for message authentication, including digital signatures. The function was developed in 1991 by Rivest.

multicast address An IP address that identifies a group of interfaces in a particular way. A packet that is sent to a multicast address is delivered to all of the interfaces in the group.

network interface card (NIC) Network adapter that is either internal or a separate card that serves as an interface to a link. The Sun Crypto Accelerator 4000 board is an NIC.

node A host or a router.

packet A group of information that is transmitted as a unit over communications lines. Contains a header plus a payload.

payload The data that is carried in a packet. The payload does not include the header information that is required to get the packet to its destination.

physical interface A node's attachment to a link. This attachment is often implemented as a device driver plus a network adapter. Some network adapters can have multiple points of attachment, for example, `qfe`. The usage of *network adapter* in this document refers to a "single point of attachment."

PKI Public key infrastructure. A system of digital certificates, Certificate Authorities, and other registration authorities that verify and authenticate the validity of each party involved in an Internet transaction.

private address An IP address that is not routable through the Internet.

public key cryptography A cryptographic system that uses two different keys. The public key is known to everyone. The private key is known only to the recipient of the message. IKE provides public keys for IPsec.

RSA A method for obtaining digital signatures and public key cryptosystems. The method was first described in 1978 by its developers, Rivest, Shamir, and Adleman.

SA See security association (SA).

SADB Security Associations Database. A table that specifies cryptographic keys and cryptographic algorithms. The keys and algorithms are used in the secure transmission of data.

security association (SA)	An association that specifies security properties from one host to a second host.
security parameter index (SPI)	An integer that specifies the row in the security associations database (SADB) that a receiver should use to decrypt a received packet.
SHA-1	Secure Hashing Algorithm. The algorithm operates on any input length less than 2^{64} to produce a message digest. The SHA-1 algorithm is input to DSA.
SPI	See security parameter index (SPI).
symmetric key cryptography	An encryption system in which the sender and receiver of a message share a single, common key. This common key is used to encrypt and decrypt the message. Symmetric keys are used to encrypt the bulk of data transmission in IPsec. DES is one example of a symmetric key system.
Triple-DES	Triple-Data Encryption Standard. A symmetric-key encryption method. Triple-DES requires a key length of 168 bits. Triple-DES is also written as 3DES.
tunnel	The path that is followed by a datagram while it is encapsulated.
virtual private network (VPN)	A single, secure, logical network that uses tunnels across a public network such as the Internet.

Index

D

E

W