IPsec and IKE Administration Guide

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

Part No: 816–7264–10
April 2003

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, SunOS, Sun ONE Certificate Server, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software–Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, SunOS, Sun ONE Certificate Server, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Adobe PostScript

030306@5533

Contents

Preface

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 system.
- You have updated the SunOS 5.9 operating system with the Solaris 9 4/03 Release.
- You have set up any networking software that you plan to use.

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

Note – The Solaris operating system runs on two types of hardware, or platforms—SPARC® and x86. The Solaris operating system 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 provides protection for IP datagrams.

Chapter 2 provides procedures for implementing IPsec on your network.

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

Chapter 4 provides procedures for implementing IKE.

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; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`**
		`Password:`

TABLE P–1 Typographic Conventions *(Continued)*

Typeface or Symbol	Meaning	Example
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

IPsec (Overview)

The IP Security Architecture (IPsec) provides cryptographic protection for IP datagrams in IPv4 and IPv6 network packets. The 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:

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.

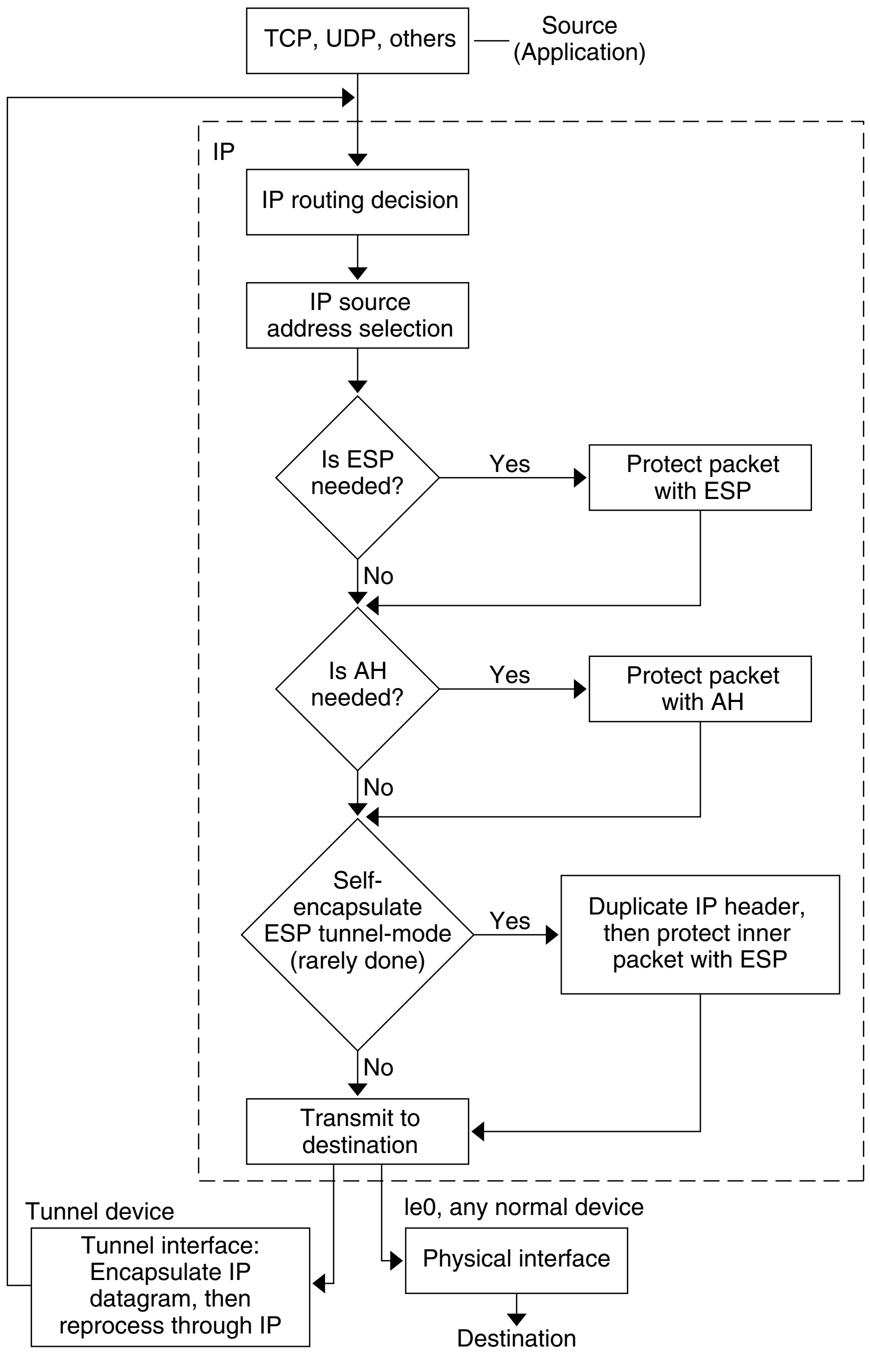


FIGURE 1–1 IPsec Applied to Outbound Packet Process

Figure 1–2 shows the IPsec inbound process.

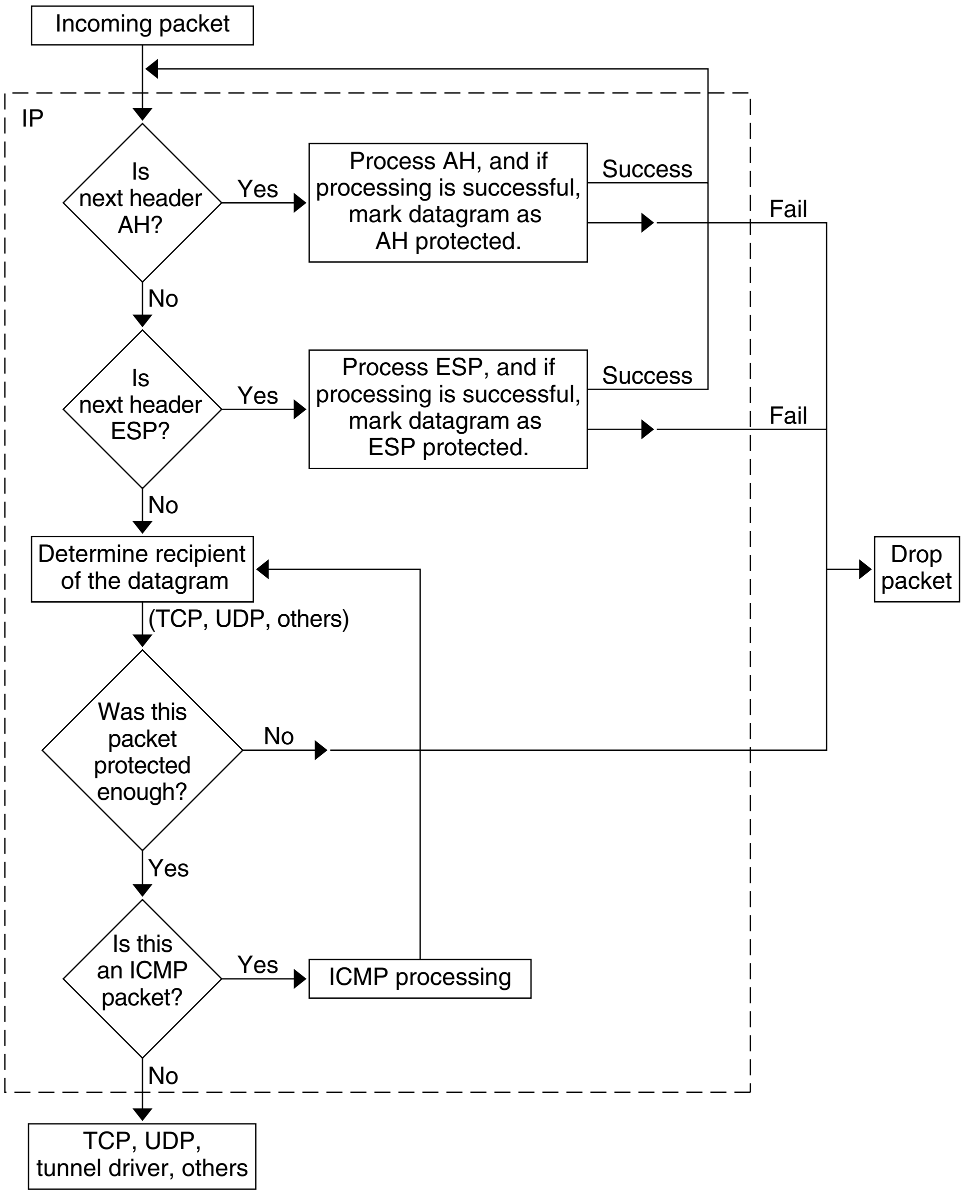

Incoming packet
IP
Is next header AH?
Yes
Process AH, and if processing is successful, mark datagram as AH protected.
Success
Fail
No
Is next header ESP?
Yes
Process ESP, and if processing is successful, mark datagram as ESP protected.
Success
Fail
No
Determine recipient of the datagram
Drop packet
(TCP, UDP, others)
Was this packet protected enough?
No
Yes
Is this an ICMP packet?
Yes
ICMP processing
No
TCP, UDP, tunnel driver, others

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 security parameter index, 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`(7P) interface, enables privileged applications to manage the database. The `in.iked`(1M) daemon provides automatic key management.

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`(1M) command. Currently, SAs on IPv4 packets can use automatic key management, while SAs on IPv6 packets require manual management.

See "IKE Overview" on page 47, for how IKE manages cryptographic keys automatically for IPv4 hosts. See "Keying Utilities" on page 23, for how the administrator can manually manage the cryptographic keys by using the `ipseckey` command.

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, a new IP 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 `ndd`(1M). 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.

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 ESP 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 `ndd`(1M). 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.

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 system. 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 system. 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

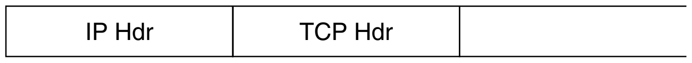
You use the `ipsecconf`(1M) command to configure system-wide policy.

IPsec applies 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.

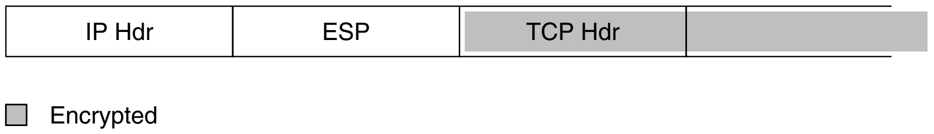
Policy that normally protects a datagram can be bypassed. You can either specify an exception in system-wide policy, or you can request a bypass in per-socket policy. For intra-system traffic, policies are enforced, but actual security mechanisms are not applied. Instead, the outbound policy on an intra-system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

Transport and Tunnel Modes

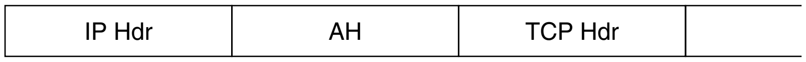
When you invoke ESP or AH after the IP header to protect a datagram, you are using transport mode. An example follows. A packet starts off with the following header:



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

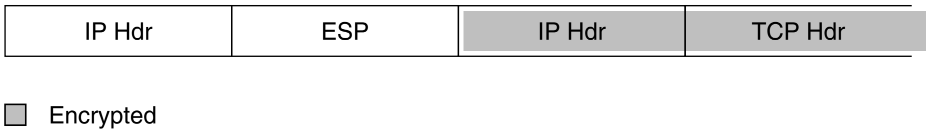


AH, in transport mode, protects the data as follows:



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:



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`(1M) 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. See the `ipsec`(7P) man page for details on per-socket policy.

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. See the `tun`(7M) man page and "Solaris Tunneling Interfaces for IPv6" in *System Administration Guide: IP Services* for more information.

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.

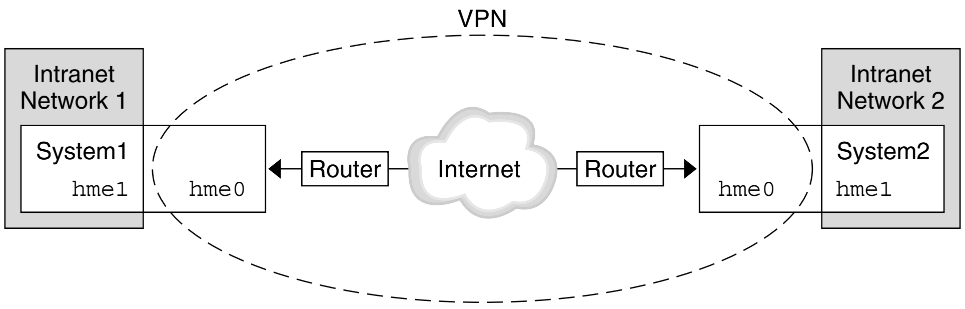


FIGURE 1–3 Virtual Private Network

See "How to Set Up a Virtual Private Network" 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 27.

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 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`(1M) starts and loads the `/etc/inet/ike/config` file. See "IKE Utilities and Files" on page 50.

IPsec Policy Command

You use the `ipsecconf`(1M) command to configure the IPsec policy for a host. When you run the command to configure 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. See the `ifconfig`(1M) and `tun`(7M) man pages about how to protect forwarded packets.

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 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 system, 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:

```
#
#ident    "@(#)ipsecinit.sample    1.6  01/10/18 SMI"
#
# Copyright (c) 1999,2001 by Sun Microsystems, Inc.
# All rights reserved.
#
# This file should be copied to /etc/inet/ipsecinit.conf to enable IPsec
# systemwide policy (and as a side-effect, load IPsec kernel modules).
# Even if this file has no entries, IPsec will be loaded if
# /etc/inet/ipsecinit.conf exists.
#
# Add entries to protect the traffic using IPsec. The entries in this
# file are currently configured using ipsecconf from inetinit script
# after /usr is mounted.
#
# For example,
#
```

```
#     {rport 23} ipsec {encr_algs des encr_auth_algs md5}
#
# Or, in the older (but still usable) syntax
#
#    {dport 23} apply {encr_algs des encr_auth_algs md5 sa shared}
#    {sport 23} permit {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}
#
# Or, in the older (but still usable) syntax
#
#    {daddr 10.5.5.0/24} apply {auth_algs any sa shared}
#    {saddr 10.5.5.0/24} permit {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.
#
#
# WARNING:    This file is read before default routes are established, and
#        before any naming services have been started. The
#        ipsecconf(1M) command attempts to resolve names, but it will
#        fail unless the machine uses files, or DNS and the DNS server
#        is reachable via routing information before ipsecconf(1M)
#        invocation.  (that is, the DNS server is on-subnet, or DHCP
#        has loaded up the default router already.)
#
#        It is suggested that for this file, use hostnames only if
#        they are in /etc/hosts, or use numeric IP addresses.
#
#        If DNS gets used, the DNS server is implicitly trusted, which
#        could lead to compromise of this machine if the DNS server
#        has been compromised.
#
```

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.

Policy cannot be changed for TCP sockets or UDP sockets on which a `connect`(3SOCKET) or `accept`(3SOCKET) has been issued. A socket whose policy cannot be changed is called a latched socket. Adding new policy entries does not affect latched sockets.

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 addresses. See Chapter 4 for how to set up IKE. The manual keying utility is the `ipseckey`(1M) command.

You use the `ipseckey` command to manually manipulate the security association databases with the `ipsecah`(7P) and `ipsecesp`(7P) protection mechanisms. You can also use the `ipseckey` command to set up security associations between communicating parties when automated key management is not available. An example is communicating parties that have IPv6 addresses.

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 IPsec policy on a tunnel interface, and the `snoop` command can parse AH and ESP headers.

`ifconfig` Command

To support IPsec, the following security options have been added to the `ifconfig`(1M) command:

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

`auth_algs`

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

`auth_algs` *authentication_algorithm*

The algorithm can be either a number or an algorithm name, including the parameter *any*, to express no specific algorithm preference. You must specify all IPsec tunnel properties on the same command line. 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`

This option enables IPsec ESP for a tunnel, with the authentication algorithm specified. 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

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

```
encr_algs encryption_algorithm
```

For the algorithm, you can specify either a number or an algorithm name. You must specify all IPsec tunnel properties on the same command line. 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*.

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

`snoop` Command

The `snoop` command can now parse AH and ESP headers. Because ESP encrypts its data, `snoop` cannot see encrypted headers that are protected by ESP. AH does not encrypt data, so traffic can still be inspected with `snoop`. 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 45.

CHAPTER 2

Administering IPsec (Task)

This chapter provides procedures for implementing IPsec on your network. The procedures are described in Table 2–1.

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

TABLE 2–1 Implementing IPsec Task Map

Task	Description	For Instructions, Go To …
Secure traffic between two IPv6 systems	Involves adding addresses to the `/etc/inet/ipnodes` file, entering IPsec policy in the `/etc/inet/ipsecinit.conf` file, manually adding keys with the `ipseckey` command, and invoking the `ipsecinit.conf` file.	"How to Secure Traffic Between Two Systems" on page 28
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, and activating the file.	"How to Secure a Web Server" on page 33

TABLE 2–1 Implementing IPsec Task Map *(Continued)*

Task	Description	For Instructions, Go To …
Set up a virtual private network	Involves turning off IP forwarding, turning on IP strict destination multihoming, disabling most network and Internet services, adding security associations, configuring IPsec policy, and configuring a secure tunnel. VPN also involves turning on IP forwarding, configuring a default route, and running the routing protocol.	"How to Set Up a Virtual Private Network" on page 35
Generate random numbers	Involves generating numbers from the Solaris `/dev/random` device.	"How to Generate Random Numbers" on page 41
Create security associations manually	Involves using the `ipseckey` command to create security associations when additional interfaces are being protected.	"How to Create IPsec Security Associations Manually" on page 42
Replace current security associations	Involves flushing current security associations before you enter new keying material.	"Example—Replacing IPsec Security Associations" on page 44
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 45

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. The system names `enigma` and `partym` are examples only. 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:

- 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

  With shared security associations, 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 total 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.**

   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
   ```

   These names are examples only. Use the names of your systems when securing traffic between your systems.

   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 `/etc/inet/ipsecinit.conf`.

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

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

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

   b. **On `partym`, 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 security associations between the two systems.**

   On each system, edit a read-only `/etc/inet/secret/ipseckeys` file. A read-only file has permissions of 400. A pair of security associations for ESP and AH protection has the following format in the `ipseckeys` file:

```
add protocol spi random-hex-string dst local-system \
    encr_alg protocol-algorithm \
    encrkey random-hex-string-of-algorithm-specified-length

add protocol spi random-hex-string dst local-system \
    auth_alg protocol-algorithm \
    authkey random-hex-string-of-algorithm-specified-length

add protocol spi random-hex-string dst remote-system \
    encr_alg protocol-algorithm \
    encrkey random-hex-string-of-algorithm-specified-length

add protocol spi random-hex-string dst remote-system \
    auth_alg protocol-algorithm \
    authkey random-hex-string-of-algorithm-specified-length
```

protocol	One of `esp` or `ah`. The `ah` protocol uses `auth_alg` and `authkey` arguments. The `esp` protocol uses `encr_alg` and `encrkey` arguments. `esp` also uses the `auth_alg` and `authkey` arguments that `ah` uses.
random-hex-string	Random number of up to eight characters in hexadecimal format. If you enter more numbers than the SPI accepts, the system ignores the extra numbers. If you enter fewer numbers than the SPI accepts, the system pads your entry.
local-system	Name of the local system
remote-system	Name of the remote system
protocol-algorithm	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	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. **Generate the random numbers.**

   You need three random numbers for outbound traffic, and three random numbers for inbound traffic. Therefore, you need the following for each system:

   - 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 41 for the procedure.

b. **For example, on `enigma`, the `ipseckeys` file might look like the following:**

```
# for inbound packets
add esp spi c83f5a4b dst enigma encr_alg 3DES \
      encrkey b6a8f89213a796bde03c601029861eae91c65783368165a6
#
add ah spi 2f526ae6 dst enigma auth_alg MD5
      authkey 305ec56369ca62c2ae804690c5713e18

# for outbound packets
add esp spi 0cecc4b2 dst partym  encr_alg 3DES \
      encrkey 802e89f9f9b929ea2b615641b71ac7034a540d3cbeeaf6a9
#
add ah spi a75bbe5f dst partym auth_alg MD5 \
      authkey 2ae8b94967e6b9b0dd16e6d4b7ea7278
```

c. **The `ipseckeys` on `partym` uses identical keys. The comments differ, because `dst enigma` is inbound on `enigma`, and outbound on `partym`:**

```
# for outbound packets
add esp spi c83f5a4b dst enigma encr_alg 3DES \
        encrkey b6a8f89213a796bde03c601029861eae91c65783368165a6
#
add ah  spi 2f526ae6 dst enigma auth_alg MD5
        authkey 305ec56369ca62c2ae804690c5713e18

# for inbound packets
add esp spi 0cecc4b2 dst partym  encr_alg 3DES \
        encrkey 802e89f9f9b929ea2b615641b71ac7034a540d3cbeeaf6a9
#
add ah  spi a75bbe5f dst partym auth_alg MD5 \
        authkey 2ae8b94967e6b9b0dd16e6d4b7ea7278
```

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

---

6. **Reboot.**

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

7. **To verify that packets are being protected, see "How to Verify That Packets are Protected" on page 45.**

Example—Securing Traffic Between IPv6 Addresses Without Rebooting

The following example describes how to test that you can secure traffic between systems with IPv6 addresses. In a production environment, to reboot is safer than to run the `ipsecconf` command.

1. Do the procedure "How to Secure Traffic Between Two Systems" on page 28 through Step 5.
2. Instead of rebooting, use the `ipseckey` command to add the security associations to the database.

   ```
   # ipseckey -f /etc/inet/secret/ipseckeys
   ```
3. Activate IPsec policy with the `ipsecconf` command:

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

   Note – Read the warning when you execute the command. A socket that is already in use (latched) provides an unsecured back door into the system.

Example—Securing Traffic Between IPv4 Addresses

The following example describes how to secure traffic between systems with IPv4 addresses. The example uses automatic key management (IKE) to create security associations. IKE requires less administrative intervention, and scales easily to secure a large amount of traffic.

1. Replace the `/etc/inet/ipnodes` file in Step 2 of the preceding task with the `/etc/hosts` file, as in the following:

   On the `partym` system, add `enigma` to the `/etc/hosts` file:

   ```
   # echo "192.168.116.16 enigma" >> /etc/hosts
   ```

   On the `enigma` system, add `partym` to the `/etc/hosts` file:

   ```
   # echo "192.168.13.213 partym" >> /etc/hosts
   ```
2. Edit the `ipsecinit.conf` file to add the IPsec policy entries as in Step 4.
3. You can create keys in one of two ways:
   - Configure IKE to generate the keys automatically. IKE also refreshes the keys automatically. To configure IKE, follow one of the configuration procedures in Table 4–1. For the syntax of the IKE configuration file, see the `ike.config`(4) man page.

You should configure IKE unless you have good reason to generate and maintain your keys manually.

- If you do not activate the IKE daemon, `in.iked`, then you can manually create the keys, as described in Step 5 in "How to Secure Traffic Between Two Systems" on page 28.

4. Reboot.

   To secure traffic without rebooting, use the `ipseckey` and `ipsecconf` commands.

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

   Note – Read the warning when you execute the command. A socket that is already in use (latched) provides an unsecured back door into the system.

   ---

5. To verify that packets are being protected, see "How to Verify That Packets are Protected" on page 45.

▼ 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 non-secured DNS client requests. All other traffic requires ESP with Blowfish and SHA-1 algorithms. Other traffic also uses a shared SA for outbound traffic. Shared SAs reduce the number of security associations 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 total 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.
       {sport 80 ulp tcp} bypass {dir out}
   ```

```
    {dport 80 ulp tcp} bypass {dir in}
    {sport 443 ulp tcp} bypass {dir out}
    {dport 443 ulp tcp} bypass {dir in}

    # Outbound DNS lookups should also be bypassed.
    {dport 53} bypass {dir out}
    {sport 53} bypass {dir in}

    # 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 the previous step.

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

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

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

```
# chmod 400 IPsecWebInitFile
```

6. **To secure the web server without rebooting, use the `ipseckey` and `ipsecconf` commands.**

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

Note – Read the warning when you execute the command. A socket that is already in use (latched) provides an unsecured back door into the system.

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

If you want a remote system to communicate securely with the Web server for non-Web traffic, their policies must match. The following policy in a remote system's `ipsecinit.conf` file enables the system to communicate with the Web server:

```
# Communicate with Web server about non-Web stuff
#
    {} permit {encr_algs blowfish encr_auth_algs sha}
    {} apply {encr_algs blowfish encr_auth_algs sha sa shared}
```

▼ How to Set Up a Virtual Private Network

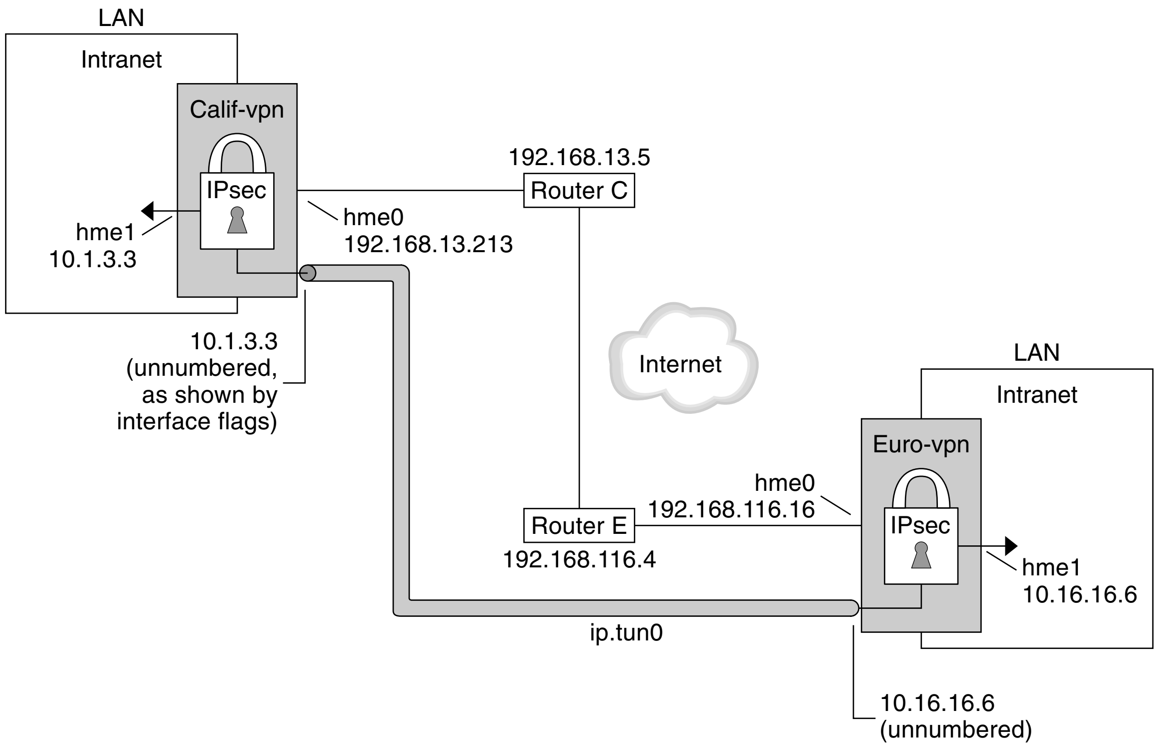
This procedure shows you how to set up a VPN by using the Internet to connect two networks within an organization. The procedure then shows you how to secure the traffic between the networks with IPsec.

This procedure extends the procedure, "How to Secure Traffic Between Two Systems" on page 28. In addition to connecting two machines, you are connecting two intranets that connect to these two machines. The machines in this procedure function as gateways.

The procedure assumes the following setup:

- Each system is using an IPv4 address space.
- 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 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.

For a description of VPNs, see "Virtual Private Networks" on page 19. 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
Host 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 total security of the system is reduced to the security of the remote login session.

2. **Turn off IP forwarding:**

   ```
   # ndd -set /dev/ip ip_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:**

   ```
   # ndd -set /dev/ip ip_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, fewer packets flow through the system. Multihoming shuts down the flow of packets except for packets that are going to system addresses. For system addresses, multihoming delivers only packets that arrive on the interface that corresponds to the destination IP address.

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

   a. **Edit the `inetd.conf` to remove all but essential services, and then type the following command:**

      ```
      # 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. Alternately, you can use techniques that are similar to the techniques in "How to Secure a Web Server" on page 33.

---

b. **If `inetd.conf` has not been edited to remove all but essential services, type the following command on a command line:**

```
# pkill inetd
```

c. **Disable other Internet services, such as SNMP, NFS, and so on, by typing one or more commands such as the following examples, as needed:**

```
# /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, telnet, or rlogin could be exploited.

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

The IKE daemon automatically creates the security associations that you configure IKE to create. You can use one of the following procedures to configure IKE for the VPN:

- "How to Configure IKE With Pre-Shared Keys" on page 58
- "How to Configure IKE With Self-Signed Public Certificates" on page 66
- "How to Configure IKE With Public Keys Signed by a Certificate Authority" on page 69

If the systems are using IPv6 addresses, you must manually create the security associations. For the steps, see "How to Create IPsec Security Associations Manually" on page 42.

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

a. **For example, on `enigma`, 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. **For example, on `partym`, 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 is part of the LAN.

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

   The entry in `ipsecinit.conf` would look like 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 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 `enigma`, 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. **For example, on `partym`, 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.**

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

   # ndd -set /dev/ip ip.tun0:ip_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. Your setting the interface's *private* flag prevents these advertisements.

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

    This 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 `enigma`, add the following route:**

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

    b. **On `partym`, 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 looks like 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 `enigma`, put its Internet router in the `/etc/defaultrouter` file:**

    ```
    # vi /etc/defaultrouter

    192.168.116.4 router-E
    ```

b. **Put `partym`'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, and thus reduce 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 `enigma`, 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 `partym`, 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`. 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
```

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

16. **On each system, run a routing protocol:**

```
# in.routed
```

▼ How to Generate Random Numbers

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

1. **Generate random keys.**

   On a Solaris system, you can use the `od` command.

```
# od -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	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 numbers 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 an SPI, you can use an 8–character hexadecimal number.

▼ How to Create IPsec Security Associations Manually

If the systems are using IPv6 addresses, you must manually create IPsec security associations.

Note – If you are running an IPv4 network, use IKE to manage security associations. For how to use IKE to manage SAs, see "Implementing IKE Task Map" on page 57.

---

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 total security of the system is reduced to the security of the remote login session.

---

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

```
# ipseckey
```

```
>
```

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

3. **To create security associations, or to replace the security associations that you just flushed, type the following command.**

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

random-hex-string	A random hexadecimal number of up to eight characters long. If you enter more numbers than the SPI accepts, the system ignores the extra numbers. If you enter fewer numbers than the SPI accepts, the system pads your entry.
protocol	One of `esp` or `ah`.
addr	The IP address of one system.
addr2	The IP address of the peer system of *addr*.
protocol-algorithm	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	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 `enigma` type the following commands to protect outbound packets. Use random numbers that you generate.**

   ```
   > add esp spi 8bcd1407 src 192.168.116.16 dst 192.168.13.213 \
   encr_alg 3DES \
   encrkey d41fb74470271826a8e7a80d343cc5aae9e2a7f05f13730d

   > add ah spi 18907dae 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` mode on `enigma`, type the following commands to protect inbound packets. Use random numbers that you generate.**

```
> add esp spi 122a43e4 src 192.168.13.213 dst 192.168.116.16 \
encr_alg 3des \
encrkey dd325c5c137fb4739a55c9b3a1747baa06359826a5e4358e

> add ah spi 91825a77 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 security association. You *should* assign different keys and a different SPI for each security association.

---

4. **Type Control-D or `quit` to exit `ipseckey` command mode.**
5. **To ensure that the keying material is available to IPsec at reboot, add the keying material to the `/etc/inet/secret/ipseckeys` file on `enigma`.**

```
add esp spi 8bcd1407 dst partym  encr_alg 3DES \
   encrkey  d41fb74470271826a8e7a80d343cc5aae9e2a7f05f13730d
#
add ah spi  18907dae  dst partym auth_alg MD5  \
   authkey  e896f8df7f78d6cab36c94ccf293f031
#
#
add esp spi 122a43e4 dst enigma encr_alg 3DES \
    encrkey 137fb4739a55c9b3a1747baa06359826a5e4358e
#
add ah spi  91825a77  dst enigma auth_alg MD5  \
   authkey  ad9ced7ad5f255c9a8605fba5eb4d2fd
```

6. **Repeat Step 1 through Step 5 on `partym`.**

   The keying material on the two systems must be identical.

Example—Replacing IPsec Security Associations

To prevent an adversary from having time to break your cryptosystem, you need to refresh your keying material. When you replace the SAs on one system, the SAs must also be replaced on the communicating system.

When replacing security associations, remove the old keys before you add new keys. Use the `flush` command in `ipseckey` command mode to remove the old keys. Then add the new keying information.

```
# ipseckey
> flush
> add esp spi ...
```

▼ 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 root or 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 root.**

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

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

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

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

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

4. **In the `snoop` window on `partym`, you should see output that looks something like 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 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 system 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 packets can take advantage of IKE.

When IKE is used on a system with a Sun™ Crypto Accelerator 1000 card, the public key operations are off-loaded to the card. Operating system resources are not used for public-key operations.

This chapter contains the following information:

IKE Overview

The Internet Key Exchange (IKE) daemon, `in.iked`(1M), negotiates and authenticates keying material for security associations in a protected manner. The daemon uses random seeds for keys from internal functions provided by the SunOS™. 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.

When the IKE daemon discovers a remote host's public encryption key, the local system can then use that key. The system encrypts messages by using the remote host's public key. The messages can be read only by that remote host. 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. An ISAKMP security association is a secure channel for IKE to negotiate keying material for the IP datagrams. Unlike IPsec SAs, the ISAKMP security associations 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 interfaces that are affected, the algorithms that are used, the authentication method, and if PFS is used. The two authentication methods are pre-shared 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 Sun™ Open Net Environment (Sun ONE) Certificate Server, Entrust, 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 hosts that are running the IKE daemon. IKE uses the secure channel that was created in Phase 1 to protect the transmission of keying material. The IKE daemon creates the keys from a random number generator by using the `/dev/random`. 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.

IKE Configuration Choices

For two IKE daemons to authenticate each other, the configuration file for IKE policy, `ike.config`(4), must be valid. Also, keying material must be available. The configuration file contains IKE policy entries. The entries determine the method for authenticating the Phase 1 exchange. The choices are pre-shared keys or public key certificates.

The key pair `auth_method preshared` indicates that pre-shared keys are used. Values for `auth_method` other than `preshared` are one indication 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.

Using Pre-Shared Keys

Pre-shared keys are created by an administrator on one system, and shared out of band with administrators of communicating systems. The administrator 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`(4) file is for IKE as the `ipseckeys` file is for IPsec. Compromise of the keys in the `ike.preshared` file compromises all keys that are derived from the keys in the file.

One system's pre-shared key must be identical to its communicating system's key. The keys are tied to a particular IP address, and are most secure when one administrator controls the communicating systems.

Using 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 method of 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 communicating system. The communicating 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 hosts.

Self-signed certificates are a halfway point between pre-shared keys and CAs. Unlike pre-shared keys, a self-signed certificate can be used on a mobile machine, or a system that might be renumbered. To self-sign a certificate, the administrator uses a DNS (`www.example.org`) or EMAIL (`root@domain.org`) alternative name.

Public keys can be delivered by a PKI or a CA organization. The public keys and their accompanying CAs are installed in the `/etc/inet/ike/publickeys` directory by the administrator. Vendors also issue certificate revocation lists (CRLs). Along with installing the keys and CAs, the administrator is 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. Like self-signed certificates, CAs can be used on a mobile machine, or on a system that might be renumbered. Unlike self-signed certificates, CAs very easily scale to protecting 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 card to handle the public key operations. For information on how to configure IKE to offload its computations to the accelerator card, see "How to Use the Sun Crypto Accelerator 1000 Card With IKE" on page 75.

IKE Utilities and Files

This section describes the configuration files for IKE policy and various commands that implement IKE. For instructions about how to implement IKE for your IPv4 network, see "Implementing IKE Task Map" on page 57.

TABLE 3–1 List of IKE Files and Commands

File or Command	Description
`in.iked`(1M) daemon	Internet Key Exchange (IKE) daemon. Activates automated key management.
`ikeadm`(1M)	IKE administration command. For viewing and modifying IKE policy.
`ikecert`(1M)	Certificate database management command. For manipulating local public-key certificate databases.
`/etc/inet/ike/config` file	Configuration file for 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.

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

File or Command	Description
`/etc/inet/secret/ike.preshared` file	Pre-shared keys file. Contains secret keying material for Phase 1 authentication. Used when configuring IKE with pre-shared keys.
`/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 to hold public keys and certificate files. Contains the public key part of a public-private key pair.
`/etc/inet/ike/crls` directory	Directory to hold revocation lists for public keys and certificate files.

IKE Daemon

The `in.iked`(1M) daemon automates the management of cryptographic keys for IPsec on a Solaris host. The daemon negotiates with a remote host 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 hosts that plan to communicate securely.

The IKE daemon is automatically loaded at boot time if the configuration file for 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 Phase 1. The peer is defined in the IKE policy file, as are the authentication methods. The daemon then establishes the keys for the session in Phase 2. 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 programs support the IKE daemon. The `ikeadm`(1M) command enables the administrator to view IKE policy. You can also use the command to modify IKE policy. The `ikecert`(1M) command enables the administrator to view and manage the public-key databases, `ike.privatekeys` and `publickeys`.

IKE Policy File

The configuration file for IKE policy, `/etc/inet/ike/config`, provides the keying material 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 hosts or networks that the keying material is for, and specifies the authentication method. See "IKE Tasks" on page 58 for examples of valid policy files. See the `ike.config`(4) man page for examples and descriptions of its parameters.

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

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 22 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
- Debug IKE processes

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

0x0, or base level	At the base level of privilege, you cannot view keying material. You also cannot modify the 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 pre-shared 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 24 for details.

Pre-Shared Keys Files

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

- The `ike.preshared` file is created by an administrator when the `ike/config` file requires pre-shared keys. The `ike.preshared` file contains keying material for ISAKMP SAs, that is, for IKE authentication. Because the pre-shared 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. For IPv6 hosts, you manually create and update the keys in this file. See "IPsec Tasks" on page 28 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.

IKE Public Key Databases and Commands

The `ikecert`(1M) command manipulates the local host's public-key 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`.

`ikecert certlocal` Command

The `certlocal` subcommand manages the private-key database in the `/etc/inet/secret/ike.privatekeys` directory. Options to the subcommand enable you to add, view, and remove private keys. The command also creates either a self-signed certificate or a certificate request. The `-ks` option creates a self-signed certificate, and the `-kc` option creates a certificate request.

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

TABLE 3–2 Correspondences Between `ike certlocal` and `ike/config` Values

`certlocal` options	`ike/config` entry	Notes
`-A` *Subject Alternate Name*	`cert_trust` *Subject Alternate Name*	A nickname that uniquely identifies the certificate. Possible values are IP address, email address, and domain name.
`-D` *X.509 Distinguished Name*	*X.509 Distinguished Name*	The full name of the certificate authority that includes Country, Organization name, Organizational Unit, and Common Name.
`-t dsa-sha1`	`auth_method dss_sig`	Slightly slower than RSA. Is not patented.
`-t rsa-md5` `-t rsa-sha1`	`auth_method rsa_sig`	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 Distinguished Name, is the biggest payload.
`-t rsa-md5` `-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.

If you issue a certificate request with the `ikecert certlocal -kc` command, you send the output of the command to a PKI organization. If your company runs its own PKI, you send the output to your PKI administrator. The organization or your PKI administrator then creates keying material. You use the keying material that is returned to you as input to the `certdb` and `certrldb` subcommands.

`ikecert certdb` Command

The `certdb` subcommand manages the public-key database, `/etc/inet/ike/publickeys`. 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 communicating system. See "How to Configure IKE With Self-Signed Public Certificates" on page 66 for the procedure. The command also accepts the certificate that you receive from a PKI or CA as input. See "How to Configure IKE With Public Keys Signed by a Certificate Authority" on page 69 for the procedure.

`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 Access a Certificate Revocation List” on page 73 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`. You use the `ikecert certdb` command to populate the directory.

The files 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 communicating 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 `ike.privatekeys` directory holds private key files that are part of a public-private key pair, keying material for ISAKMP SAs. The directory is protected at `0700`. The private key in this database must have a public key counterpart in the `publickeys` database.The `ikecert certlocal` command populates this directory. Private keys are not effective until their public key counterparts, self-signed certificates or CAs, are installed in the `/etc/inet/ike/publickeys` directory.

`/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

Internet Key Exchange (Task)

This chapter provides procedures for implementing IKE. IKE, after it is configured for your systems, automatically generates keying material for IPsec on your network. The procedures are described in the Table 4–1.

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.

Implementing IKE Task Map

TABLE 4–1 Implementing IKE Task Map

Task	Description	For Instructions, Go To …
Configure IKE with pre-shared 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 Pre-Shared Keys" on page 58
Refresh pre-shared keys on a running IKE system	Involves checking the IKE privilege level and editing the `ipseckeys` file with fresh keying material on communicating systems.	"How to Refresh Existing Pre-Shared Keys" on page 61
Add pre-shared 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 Pre-Shared Key" on page 62

TABLE 4–1 Implementing IKE Task Map *(Continued)*

Task	Description	For Instructions, Go To …
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 communicating system with the `ikecert certdb` command.	"How to Configure IKE With Self-Signed Public Certificates" on page 66
Configure IKE with a PKI Certificate Authority	Involves sending output from the `ikecert certlocal -kc` command to a PKI organization, and installing the public key, CA, and CRL from the organization.	"How to Configure IKE With Public Keys Signed by a Certificate Authority" on page 69
Update the CA revocation lists	Involves accessing a PKI organization's CRL from a central distribution point.	"How to Access a Certificate Revocation List" on page 73
Use the Sun Crypto Accelerator 1000 card with IKE	Involves setting the path to the PKCS#11 library for the device.	"How to Use the Sun Crypto Accelerator 1000 Card With IKE" on page 75

IKE Tasks

This section provides procedures that enable you to automatically manage the keys that secure traffic between two systems with IPv4 addresses. 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.

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

▼ How to Configure IKE With Pre-Shared Keys

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 total 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 `/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 28.

   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"
  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 `auth_method` must be on the same line.

   b. **Modify the 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"
  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
}
```

Note – These system names are examples only. Use the names and addresses of your systems when securing traffic between your systems.

---

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

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

5. **Generate random keys.**

   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 command, see "How to Generate Random Numbers" on page 41 and the `od`(1) man page.

6. **Create the file `/etc/inet/secret/ike.preshared` on each system. Put the pre-shared key in each file.**

   The authentication algorithm in this example 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 pre-shared key. The output of the MD5 algorithm is 128 bits, or 32 characters. Since a longer key length is a good idea, the example key is 56 characters long.

   a. **For example, on the `enigma` system, `ike.preshared` would look like 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, `ike.preshared` would look like 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 pre-shared keys must be identical.

IKE is now configured for use with IPsec.

Example—Check That the Pre-Shared Keys are Identical

If the pre-shared keys on the communicating systems are not identical, you get the following error message:

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

To view the pre-shared key, the `in.iked` daemon must be running at privilege level 0x2. On each system, use the `ikeadm` command to dump the pre-shared key information:

```
# /usr/sbin/ikeadm get priv
Current privilege level is 0x2, access to keying material enabled
# ikeadm dump preshared
PSKEY: Pre-shared 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).
```

Compare the two dumps. If the pre-shared keys are not identical, replace one key with the other key in the `/etc/inet/secret/ike.preshared` file.

▼ How to Refresh Existing Pre-Shared Keys

This procedure assumes that you want to replace an existing pre-shared 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 total security of the system is reduced to the security of the remote login session.

2. **Generate random keys and choose one of the keys.**

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
         03efe016 216e60ac e316f663 a2f073e0
         7f90d069 316d99b5 00f8384c 2142610a
```

For an explanation of the command, see "How to Generate Random Numbers" on page 41 and the `od`(1) man page.

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.**

   For example,

   ```
   # 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.**

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

   For example,

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

▼ How to Add a New Pre-Shared Key

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

- The `in.iked` daemon is running
- The interface that you want to protect with IPsec is an entry in the `/etc/hosts` file on both systems, for example:

  ```
  192.168.15.7 ada
  ```

- You have added a new policy entry to the `/etc/inet/ipsecinit.conf` file on both systems. For example, the entry on `enigma` looks something like the following:

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

  For example, the entry on `ada` looks something like the following:

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

- You have created a rule for the interface on `ada` in the `/etc/inet/ike/config` file on both systems. For example, the rule on `enigma` looks something like the following:

  ```
  ### ike/config file on enigma, 192.168.116.16
  ...
  ## The rule to communicate with partym-15
  { label "Enigma-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
      }
  ```

  For example, the rule on `ada` looks something like 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
  }
  ```

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

  ---

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 total 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 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.

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

   For example,

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

4. **Generate random keys and combine the output to create a key of 64 to 448 bits.**

   On a Solaris system, you can use the `od` command.

   ```
   # od -X -A n /dev/random | head -4
           0fb834c5 8d1fb4ee 500e2bea 071deb2e
           781cb483 74411af5 a9671714 672bb174
           9ad9364d 53574f27 4aacea56 c34861bb
           b4509514 145c1845 f857ff2b 6e5e3766
   ```

   For an explanation of the command, see "How to Generate Random Numbers" on page 41 and the `od`(1) man page.

5. **By some means, send the key to the administrator of the communicating system.**

   You are both going to add the same pre-shared key at the same time.

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

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

id-type	The type of the *id*.
id	IP address when *id-type* is IP.
mode	The IKE mode. `main` is the only accepted value.
key	The pre-shared key in hexadecimal format.

   For example, on host `enigma`, you add the key for the new interface, `ada`, 192.168.15.7

   ```
   # 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.
   ```

On host ada, the administrator would add the identical key, as in:

```
# 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.
```

Note – A message of the form `Error: invalid preshared key definition` indicates that you gave incorrect arguments to the `add preshared` command. You might have mistyped a parameter. You might have omitted a parameter. Retype the command correctly to add the 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
   ```

   Note – Read the warning when you execute the command. A socket that is already latched, that is, the socket is in use, provides an unsecured back door into the system.

   ---

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

    A sample of the new rules for ada and enigma are 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.

    ```
    # ikeadm read rules
    ```

11. **To ensure that IKE pre-shared keys are available at reboot, edit the `/etc/inet/secret/ike.preshared` file.**

    Enter the arguments to the add preshared command into the file on each system, as shown in the following substeps.

    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 04413a3e68854b732742024d19995f7972136a2f33e5d302bdd7b2624e4c6429
    }
```

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

```
# ike.preshared for the ada interface, 192.168.15.7
#...
{ 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 04413a3e68854b732742024d19995f7972136a2f33e5d302bdd7b2624e4c6429
    }
```

▼ How to Configure IKE With Self-Signed Public 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 total security of the system is reduced to the security of the remote login session.

---

2. **Use the `ikecert certlocal -ks` command to add a self-signed certificate to the `ike.privatekeys` database.**

```
# ikecert certlocal -ks -m keysize -t keytype \
-D dname -A altname[ ... ] [-f output]
```

`-ks`	Creates a self-signed certificate
keysize	The size of the key. *keysize* can be 512, 1024, 2048, 3072, or 4096.
keytype	The type of algorithm to use. *keytype* can be rsa-sha1, rsa-md5, or dsa-sha1.
dname	The X.509 distinguished name for the certificate subject. *dname* typically has the form: C=country, O=organization, OU=organizational unit, CN=common name. Valid tags are C, O, OU, and CN.
altname	The alternate name for the certificate. *altname* should be in the form of `tag=value`. Valid tags are `IP`, `DNS`, `EMAIL`, `URI`, `DN`, and `RID`.

output The format of the encoding output. The possible values are `pem` for PEM Base64, and `ber` for ASN.1 BER. If `-f` is not specified, `pem` is assumed.

For example,

```
# ikecert certlocal -ks -m 1024 -t rsa-md5 \
> -D "C=US, O=ExampleCompany, OU=US-Example, CN=Example" \
> -A IP=192.168.116.16
Generating, please wait...
Certificate:
Certificate generated.
Certificate added to database.
-----BEGIN X509 CERTIFICATE-----
MIICLTCCAZagAwIBAgIBATANBgkqhkiG9w0BAQQFADBNMQswCQYDVQQGEwJVUzEX
...
6sKTxpg4GP3GkQGcd0r1rhW/3yaWBkDwOdFCqEUyffzU
-----END X509 CERTIFICATE-----
```

3. **Send the certificate to the communicating system's administrator.**

   You can cut-and-paste the certificate into an email, as in:

```
To: root@us.example.com
From: root@un.example.com
Message: -----BEGIN X509 CERTIFICATE-----
MIICLTCCAZagAwIBAgIBATANBgkqhkiG9w0BAQQFADBNMQswCQYDVQQGEwJVUzEX
...
6sKTxpg4GP3GkQGcd0r1rhW/3yaWBkDwOdFCqEUyffzU
-----END X509 CERTIFICATE-----
```

4. **On the sending system, edit the `/etc/inet/ike/config` file to recognize the public keys from a communicating system. For example,**

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

cert_trust "192.168.116.16"
cert_trust "192.168.13.213"

## 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 "UN-Example to US-Example"
 local_id_type dn
 local_id "C=US, O=ExampleCompany, OU=UN-Example, CN=Example"
 remote_id "C=US, O=ExampleCompany, OU=US-Example, CN=Example"

 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 }
}
```

5. **Do the following substeps to add the communicating system's public key.**

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

   b. **Type the `ikecert certdb -a` command and type <Return>.**

      No prompts display when you type the <Return> key.

      ```
      # ikecert certdb -a <Type the Return key>
      ```

   c. **Paste the public key. Then type <Return>.**

      ```
      -----BEGIN X509 CERTIFICATE-----
      MIICL...
      ...
      KgDid/nxWPlWQU5vMAiwJXfa0sw/A12w448JVkVmEWaf
      -----END X509 CERTIFICATE-----<Type the Return key>
      ```

   d. **End the entry by typing <Control-D>.**

      ```
      <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 on one system should be identical to the public key hash on the communicating system.

   a. **For example, on `enigma`, type the `ikecert certdb -l` command.**

      ```
      enigma # ikecert certdb -l
              Certificate Slot Name: 0   Type: if-modn
              Subject Name: <C=US, O=ExampleCo, OU=UN-Example, CN=Example>
              Key Size: 1024
              Public key hash: 2239A6A127F88EE0CB40F7C24A65B818
      ```

   b. **On `partym`, type the`ikecert certlocal -l` command.**

      ```
      partym # ikecert certlocal -l
      Local ID Slot Name: 1   Type: if-modn
              Key Size: 1024
              Public key hash: 2239A6A127F88EE0CB40F7C24A65B818
      ```

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

▼ How to Configure IKE With Public Keys Signed by a Certificate Authority

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 total 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.**

   The arguments to the command should be the same arguments that you would use for a `ikecert certlocal -ks` command.

```
# ikecert certlocal -kc -m keysize -t keytype \
-D dname -A altname[ ... ] [-f output]
```

`-kc`	Creates a public and private key pair. Also, `-kc` creates a certificate request that is based on the public key that was generated.
keysize	The size of the key. *keysize* can be 512, 1024, 2048, 3072, or 4096.
keytype	The type of algorithm to use. *keytype* can be rsa-sha1, rsa-md5, or dsa-sha1.
dname	The X.509 distinguished name for the certificate subject. *dname* typically has the form: C=country, O=organization, OU=organizational unit, CN=common name. Valid tags are C, O, OU, and CN.
altname	The alternate name for the certificate. *altname* should be in the form of `tag=value`. Valid tags are `IP`, `DNS`, `EMAIL`, `URI`, `DN`, and `RID`.
output	The format of the encoding output. The possible values are `pem` for PEM Base64, and `ber` for ASN.1 BER. If `-f` is not specified, `pem` is assumed.

   For example, the following command creates a certificate request:

```
# ikecert certlocal -kc -m 1024 -t rsa-md5 \
> -D "C=US, O=ExampleCompany\, Inc., OU=US-Example, CN=Example" \
> -A "DN=C=US, O=ExampleCompany\, Inc., OU=US-Example"
Generating, please wait...
Certificate request generated.
-----BEGIN CERTIFICATE REQUEST-----
MIIByjCCATMCAQAwUzELMAkGA1UEBhMCVVMxHTAbBgNVBAoTFEV4YW1wbGVDb21w
...
lcM+tw0ThRrfuJX9t/Qa1R/KxRlMA3zckO80mO9X
-----END CERTIFICATE REQUEST-----
```

3. **Submit the certificate request to an organization that handles PKI.**

The organization might be an outside Certificate Authority or PKI. Also, a company might run its own PKI. The PKI 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 two or three certificate objects:

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

  When a URI for the CRL is embedded in the publickeys certificate, IKE can automatically retrieve the CRL for you. Similarly, when a DN entry is embedded in the publickeys certificate, IKE can automatically retrieve the CRL from an LDAP server that you specify.

  See "How to Access a Certificate Revocation List" on page 73 for an example of an embedded URI and an embedded DN entry in a publickeys certificate.

4. **Enter each certificate as the argument to one of three `ikecert` commands.**

   The commands use the `-a` option. The `-a` option adds the pasted object to the appropriate certificate database on your system. For more information, see "Using Public Key Certificates" on page 49.

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

   b. **Type the `ikecert certdb -a` command and type <Return>.**

   ```
   # ikecert certdb -a <Type the Return key>
   ```

   c. **Paste the publickeys certificate that you received from the organization and type <Return>.**

   ```
   -----BEGIN X509 CERTIFICATE-----
   ...
   -----END X509 CERTIFICATE----<Type the Return key>
   ```

   d. **End the entry by typing <Control-D>.**

   ```
   <Control-D>
   ```

   e. **Type the `ikecert certdb -a` command and type <Return>.**

   ```
   # ikecert certdb -a <Type the Return key>
   ```

   f. **Paste the organization's CA. Type <Return>. Then type <Control-D> to end the entry.**

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

g. **If the organization has sent a CRL object, type the `ikecert certrldb -a` command and type <Return>.**

```
# ikecert certrldb -a <Type the Return key>
```

h. **Paste the organization's CRL. Type <Return>. Then type <Control-D> to end the entry.**

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

   Use the name that the organization tells you to use. For example,

```
# 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-Example to UN-Example - Example PKI"
 local_id_type dn
 local_id  "C=US, O=ExampleCompany, OU=US-Example, CN=Example"
 remote_id "C=US, O=ExampleCompany, OU=UN-Example, CN=Example"

 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 }
}
```

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

---

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

   `ignore_crls` tells IKE not to search for CRLs. For example,

```
# Trusted root cert
...
cert_root "C=US, O=ExamplePKI\, Inc., OU=PKI-Example, CN=Example PKI"
ignore_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 Access a Certificate Revocation List" on page 73 for examples.

8. **Repeat Step 1 through Step 7 on the communicating system.**

   Following the example, the "…OU=UN-Example…" system runs the `ikecert` commands as you have done. The `/etc/inet/ike/config` file of the "…OU=UN-Example…" system uses its local information for local parameters. The system uses your system's information for the remote parameters.

   For example,

```
# Trusted root cert
# This certificate is from Example PKI

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

## 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 "UN-Example to US-Example - Example PKI"
 local_id_type dn
 local_id "C=US, O=ExampleCompany, OU=UN-Example, CN=Example"
 remote_id "C=US, O=ExampleCompany, OU=US-Example, CN=Example"

 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 }
}
```

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

Note – The following substeps are necessary only if the `p1_xform` in the `/etc/inet/ike/config` file uses the `rsa_encrypt` authentication method.

---

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

      You can cut-and-paste the certificate into an email, as in:

```
To: root@un.example.com
From: root@us.example.com
Message: -----BEGIN CERTIFICATE REQUEST-----
MIIByjCCATMCAQAwUzELMAkGA1UEBhMCVVMxHTAbBgNVBAoTFEV4YW1wbGVDb21w
```

```
...
lcM+tw0ThRrfuJX9t/Qa1R/KxRlMA3zckO80mO9X
-----END CERTIFICATE REQUEST-----
```

b. **On each system, add its peer's certificate to the local publickeys database.**

   For example, on the `un.example.com` system, type the following and paste the contents of the email:

```
# ikecert certdb -a <Type the Return key>
-----BEGIN CERTIFICATE REQUEST-----
MIIByjCCATMCAQAwUzELMAkGA1UEBhMCVVMxHTAbBgNVBAoTFEV4YW1wbGVDb21w
...
lcM+tw0ThRrfuJX9t/Qa1R/KxRlMA3zckO80mO9X
-----END CERTIFICATE REQUEST-----
```

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, the method requires that the IKE peers know each other's public keys. Therefore, when you use `auth_method 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. The database holds your publickeys certificate, the CA certificate, *and* the peer's certificate. The CRL database still holds any revoked certificates.

The IKE daemon authenticates itself with the public keys and with the CA when the following completes:

a. The `/etc/hosts` file on each system has been modified to include the protected interfaces

b. The `/etc/inet/ipsecinit.conf` file on each system has been modified to include the protected interfaces

c. Both systems are rebooted.

▼ How to Access a Certificate Revocation List

A Certificate Revocation List (CRL) handles 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 was shown in "How to Configure IKE With Public Keys Signed by a Certificate Authority" on page 69.
- IKE can access the CRLs from a URI whose address is embedded in the publickeys certificate from the CA.
- IKE can access the CRLs from an LDAP server whose DN entry is embedded in the publickeys certificate from the CA. You specify the LDAP server as an argument to the `ldap-list` keyword in the `/etc/inet/ike/config` file.
- 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 PKI organization by using the `ikecert certdb -lv` *certspec* command.**

`-l`	Lists certificates in the IKE certificate database.
`-v`	Lists the certificates in verbose mode. Use this option with care.
certspec	Match the *certspec* pattern to a pattern in the certificate.

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: if-modn
   (Private key in certlocal slot 0)
 Subject Name: <O=Sun Microsystems Inc, CN=example-protect.sun.com>
 Issuer Name: <CN=Sun Microsystems Inc CA (Class 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 (Class 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 Certificate Revocation List 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, put the keyword `use_http` in the host's `/etc/inet/ike/config` file.**

For example, the `ike/config` file would look like the following:

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

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

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

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

3. **To use LDAP, enter 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 look something like 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

The example shows how to use a CRL that is *not* available from a central distribution point.

If your organization's certificate does not contain a central distribution point, you can add the organization's CRL manually to the local `crls` database. You follow the organization's instructions for extracting the CRL, then add the CRL to the database with the `ikecert certrldb -a` command.

```
# ikecert certrldb -a<Type the Return key>
<Paste the CRL from the PKI organization.>

<Type the Return key.>
<Type <Control-D> to enter the CRL into the database.>
```

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

Note – The following procedure assumes that a Sun Crypto Accelerator 1000 card is attached to the system. The procedure also assumes that the software for the card 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 total 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 pathname 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 1000 card. When the card 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 Phase 1 data. Keys that are accelerated by PKCS #11 are part of Phase 1 data.

Glossary

This glossary contains definitions of network security terms.

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.
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.
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 header	An extension header that provides integrity and confidentiality to datagrams.
encapsulation	The process of a header and 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 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 associations.
IP in IP encapsulation	The mechanism for tunneling IP packets within IP packets.
IPsec	The 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 associations.
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.
node	A host or a router.
packet	A group of information that is transmitted as a unit over communications lines. Contains a header plus payload.

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.
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	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. It is input to DSA.
SPI	Security Parameters Index. An integer that specifies the row in the SADB that a receiver should use to decrypt a received packet.
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 which provides a key length of 168 bits.
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