IPSec-IKEv2 Technical Reference

Interniche Legacy Document

Version 1.00

Date: 17-May-2017 17:22

Table of Contents

1 Introduction

This manual describes InterNiche's IPSec and IKE and is intended for both the developer who wants to understand its internal architecture and for the application developer desiring only to know how to build, configure and use it within an embedded device. InterNiche's IPSec, when used with Kerberos or Pre-Shared Keys does not require IKE. Additionally, if only one of IKEv1 or IKEv2 are required then the other can be removed at compile-time.

2 Overview

2.1 What IPSec Is

InterNiche's IPSec is a software module that provides bulk encryption and authentication between endpoints and gateways over IPv4 and IPv6. It implements the Authentication Header (AH) and Encapsulating Security Protocol (ESP) of the IETF's security framework. Confirguration information and keys are provided either manually (via CLI pr APIs) or through IKE.

2.2 What IKE Is

The Internet Key Exchange (IKE) protocol is used for key agreement and key management within IPSec. It provides a scalable method of generating security associations for IPSec protocols. The protocol includes system authentication, key agreement and security association generation. This protocol is specified by the IETF standard RFC 2409 and RFC 4306 (IKEv2). Several other RFCs specify related standards.

InterNiche IKE is tightly coupled with InterNiche IPSec. While it is possible to use InterNiche IPSec without IKE, it is not possible or useful to install IKE without IPSec.

Do I need it?

Data is authenticated and secured (encrypted) using IPSEC not IKE. IPSEC can be configured without IKE however IKE enables more dynamic configuration and key management. IKE can negotiate which authentication and encryption algorithms to use and can generate keys dynamically. IKE therefore provides more flexibility, and perhaps greater security through the use of dynamic keys.

IKEv1 vs IKEv2

IKEv1 is incompatible with IKEv2. InterNiche’s IKE module supports both versions simultaneously, that is, it can be configured to accept connections initiated from a remote system and respond to whichever version is initiated. The IKE module can be configured to initiate connections to a remote system using the specified version; each security association can use a different version depending upon the needs of the remote system.

IKEv2 is more flexible than IKEv1. It is capable of negotiating more parameters such as the IKE SA (Security Association) lifetime. With IKEv1 these parameters must match exactly on each side of the connenction.

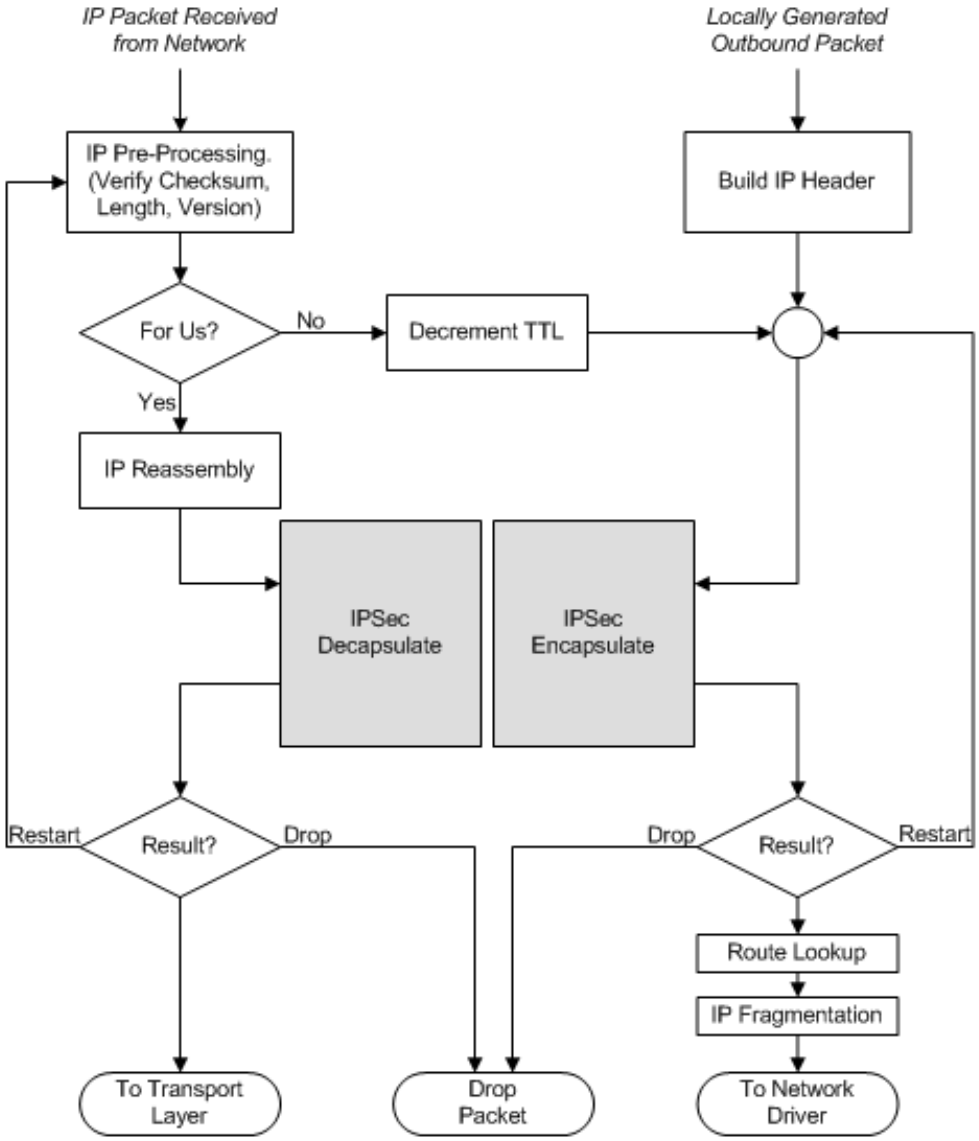
3 IPSec

3.1 Product Architecture, Packet Flow

The IPSec toolkit is designed to be modular to allow for easy replacement of any of its modules with silicon for very high performance applications. The product provides the following features and functions:

1. Packet Interface
2. Packet Classification
3. Encapsulation and De-encapsulation Engine
4. Cryptographic Engine
5. Cryptographic Library
6. Management Interface
7. Security Association and Security Policy Databases

The following flowchart shows a typical IP protocol stack processing. Integrating The IPSec Toolkit into such a protocol stack involves adding hooks in a couple of locations within the stack as shown in the flowchart.



As the flowchart shows, IPSec encapsulation is performed for outbound traffic by calling the PacketEncapsulate function. Similarly, IPSec decapsulation is performed for inbound traffic that is addressed to the system by calling the PacketDecapsulate function. The output packet from these functions is then processed as specified by the result code.

3.2 Packet Interface

This module provides an interface between IPSec and the IP protocol stack. Packets are sent to this module from the IP protocol stack and not directly from the network interface. All IP Packets received and transmitted by the IP protocol stack are sent to the IPSec module using this interface.

Since the IPSec module interfaces to the IP protocol stack, source code access to the IP protocol stack is required. For most network equipment vendors this is not a problem. If source code access is not available for the IP stack, then additional work needs to be performed before calling the IPSec toolkit. This includes typical handling of IP packets by the IP stack, the most important being the handling of fragmentation and reassembly of IP packets and IP option processing.

There are three types of packets that an IP stack handles: (a) received network packets addressed to the system, (b) received network packets that need to be routed and (c) outgoing packets that originated on the system.

The processing that IP protocol stack needs to perform before calling IPSec depends on the type of the packet:

- Received packets addressed to this system.
  Packets received from the network are processed by the IP. Incoming packets are first checked for IP header checksum validation, and checking if the packet is targeted for this system based on the IP destination address. If the destination address matches the IP address of the interface, or is a broadcast or multicast address, then the packet is for this system. On many systems, the IP destination address is checked against all IP addresses that are on this system, not just the address of the interface on which the packet arrived. If the packet is for this system, then the IP stack may perform additional processing such as IP fragment reassembly and IP options processing. After this processing but before passing the packet up to the higher layer protocol based on the protocol field of the IP header, IPSec needs to be called to decapsulate a packet if it is IPSec encapsulated and to verify security policy on the packet.
- Received packets not addressed to this system.
  If it is determined that an incoming packet is not targeted for this system and needs to be routed, then the IP stack decrements the hop count in the IP header. After decrementing the hop count, but before looking up the route to the destination, IPSec encapsulation function must be called to check the security policy and to encapsulate the packet if necessary.
- Outgoing packets that originated on the system.
  All outgoing packets that originate on this system must be passed to the IPSec module for policy checking and possible encapsulation. IPSec encapsulation function is called after the IP header is pre-pended to the packet and is properly filled in.

3.3 Packet Classification

The packet classification process extracts various fields from the packet being processed, and compares them to corresponding fields in entries in the IPsec security policy database (SPD). The selectors in a policy entry include source IP address, destination IP address, protocol, source port, and destination port. IP addresses can be specified as individual addresses or as a group of addresses. The latter can be specified via a subnet mask, range, or wild card ("any" or "any6"). The protocol and port values can be specified as zero (wildcard), or set to a specific value.

3.4 Encapsulation - De-encapsulation Engine

After a packet is classified and its security association exists in the SAD table, the packet is sent to the encapsulation/de-encapsulation module. For encapsulation, this module builds the IPSec headers and tunnel header before sending the packet to the cryptographic module. For de-capsulation, the packet is prepared for the cryptographic engine. There are several fields in the packet headers that need to be manipulated such as the mutable fields of AH and IP headers that need to be saved and zeroed before the packet is sent to the cryptographic module.

Some IPSec hardware accelerators include capability to process IPSec headers and can perform the IPSec encapsulation/de-capsulation operations. When the software is used with such hardware, then this module will be replaced with one specific to the hardware.

3.5 Management Interface

This module provides an interface to IPSec management packets. The management packets are typically sent from the IPSec management application or from the Internet Key Exchange (IKE) application. In addition, applications can register to receive asynchronous events from the IPSec modules. For example, IKE registers to receive events from the IPSec engine.

Using the management interface, security policies (SP) and security associations (SA) can be added, looked up, or deleted from the IPSec engine.

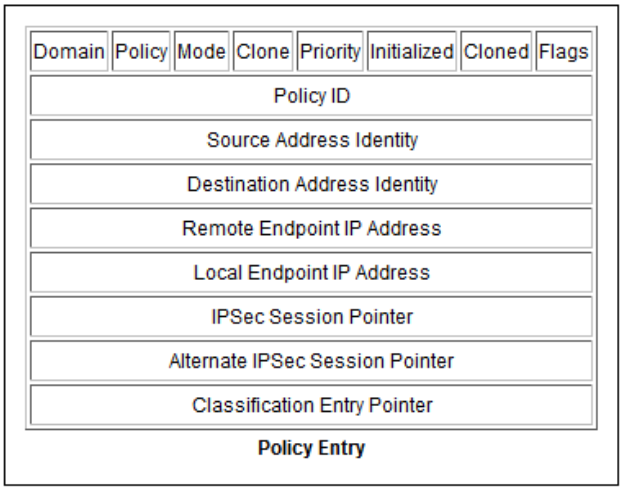
3.6 Security Policy Database

Security policies are stored in the security policy database (SPD). Each policy is identified by a security identity (id). A policy entry contains the policy's traffic selector as well as other properties of that policy.

A classification rule is added to the classification tree for each selector in the policy. The leaf entry of this branch of the classification tree points to the policy entry. The policy entry contains a back pointer to the classification tree that is useful when deleting the policy entry.

The policy entry also contains a pointer to corresponding entry in the security association database (SAD). An alternate SA pointer is also included when more than one set of SAs exists in the SAD. This can happen during the re-keying of security associations.

The table shows the fields of the policy entry:



The user specifies most of the fields in the policy entry when the policy is added to the SPD. The policy ID can either be generated by the IPSec module or can be specified by the user.

Each Policy entry that is used for securing traffic has a corresponding Session entry. The Session entry contains all information related to a single IPSec session and includes SAs for all protocols (AH, ESP) for both inbound and outbound directions.

Policies that are for bypassing IPSec or for dropping packets do not need a Session entry.

3.7 Security Association Database - SAD

Security associations are stored in the security association database (SAD). All security associations for a given IPSec session are grouped together into a single entry called IPSecSession. Each session entry in the SAD has a corresponding policy entry in the security policy database SPD.

The IPSec session for an outbound packet (unsecured) can be located by using the classification engine to locate the policy. If an SA exists for the policy, the policy entry will have a pointer to the IPSec session entry.

The SPI from either AH or ESP header of an inbound (secured) packet can be used to locate the IPSec session for the packet. The SPI contains the index in the session table. Thus the session entry can be quickly identified.

3.8 Security Policy Options

Several options are available in the policy entry to control the handling of IPSec packets. Some of these are listed here.

- Policy Action (Secure, Bypass, Drop)
  A policy entry specifies the action to perform when a network packet matches the policy. The options are (i) Secure the packet using IPSec, (ii) bypass IPSec and (iii) drop the packet.
- Transport or Tunnel Mode
  If a packet is to be secured, then it's mode must be specified. The options are (i) Transport mode and (ii) Tunnel mode.
- Clone policy
  The cloning option allows using different security associations and security keys for a policy based on any IPSec selector value. In effect, new policies are dynamically added when traffic from different hosts matches this policy.
- Explicit Congestion Notification (ECN)
  This option controls the operation of the ECN and CE bits of the TOS field of IP header. If this option is set, then full functionality ECN, as defined in the specifications, is performed by the IPSec engine.
- Don't Fragment (DF)
  This option controls how to handle the DF flag of the IP header when entering or exiting an IPSec tunnel. The options are (i) Set DF, (ii) Clear DF, or (iii) Copy DF from inner IP header.
- Replay Check
  This option enables replay checking on incoming AH or ESP packets.
- ESP Pad Check
  This option enables verification of the pad data on incoming ESP packets.
- SP_TX_DUMMY_PKT
  This option enables the transmission of dummy packets.
- SP_TX_TFC_PADDING
  This option enables the addition TFC padding to outgoing packets.
- SP_AUTH_SHA2_ICV_SHORT
  This option enables the use of 96-bit ICV instead of the RFC4868 specified length (for SHA2-xxx family of algorithms).

3.9 Dummy Packets

The IPsec module allows the transmission and reception of dummy packets (in support of traffic flow confidentiality) as outlined in Section 2.6 of RFC 4303 ("IP Encapsulating Security Payload (ESP)").

The size and inter-packet interval for transmitted dummy packets are determined randomly. These are controlled via the following in `ipsec/timer.c`:

```
#define MAX_DUMMY_PKT_PAYLOAD_SIZE 16 /* bytes */
#define MAX_DUMMY_PKT_TX_INTERVAL  30 /* seconds */
```

The above values allow the payload of a dummy packet to vary between 0 and 15 bytes, and the inter-packet interval to vary between 0 and 30 seconds. The actual size of the dummy packet is larger, since it includes an IPv4 or IPv6 header, and additional security-related headers (e.g., ESP or AH+ESP).

Dummy packets are transmitted using **ipsec_tx_dummy_pkt()** in **ipsec/timer.c**. The dummy packet generated is an IPv4 (or IPv6) packet that has a Protocol (or Next Header) field of 59 (0x3b). This allows the egress path encapsulation code to set the Next Header field in the ESP trailer appropriately. A non-dummy packet would have this field set to 4 (or 41) for IPv4 (or IPv6) (when operating in tunnel mode).

Dummy packets are only transmitted on security associations that are configured to use ESP in tunnel mode. (The security association can also use AH in addition to ESP.)

The dummy packet transmission function invokes the IPv4 (or IPv6, as the case may be) send function to perform egress path processing (e.g., fragmentation); however, it instructs those functions to skip invoking IPSec for the dummy packet.

Dummy packets are always processed (upon reception), and the data in them (extracted after they are decapsulated) is discarded.

The contents of a received dummy packet can be examined by adding debug code in **DecapsulateCryptoDone()** in **ipsec/ipsec.c**.

The number of dummy packets transmitted and received on a particular security association can be determined by entering the "`ipsec netstat -a`" CLI command.

3.10 TFC Padding

The IPsec/IKE module supports the Traffic Flow Confidentiality (TFC) Padding feature as outlined in Section 2.7 of "IP Encapsulating Security Payload (ESP)" [RFC 4303].

The policy flag parameter ('`flags`' for `IPSecMgmtAddPolicy()`, `IPSecAdminAddPolicy()`, and `IPSecAdminAddManualSA2()`. '`policy_flags`' for `IkeAdminAddIPSecConf ()`) can be used to enable or disable the inclusion of of TFC padding on outgoing packets via the newly defined `SP_TX_TFC_PADDING` bit.

The TFC padding feature should only be enabled on policies where it can be correctly supported. Sections 2.7 and 8 of RFC 4303 provides guidance on the IPSec modes (e.g., transport, tunnel) and protocols (e.g., UDP) where TFC padding can be safely enabled.

The amount of TFC padding added to an outgoing packet is determined via a call to `ipsec_compute_tfc_pad_len()` in `ipsec/process.c`. This function is programmed to return a random number between 0 and `(MAX_TFC_PADDING_SIZE - 1)` bytes as the amount of padding that needs to be added. (`MAX_TFC_PADDING_SIZE` is defined in `ipsec/ipsecapi.h` as 256 bytes.) The actual process of inserting the TFC padding occurs in `PostClassify-Encapsulate()` in `ipsec/process.c`. In the event that the TFC padding cannot be inserted into the outgoing packet, the latter is transmitted without any padding.

The IPSec module will always process received packets that contain TFC padding. The trailing padding is stripped off by the appropriate layer (e.g., IP, UDP) of the protocol stack. The amount of padding that needs to be removed can be determined from the length fields in the protocol (e.g., IP, UDP) header.

3.11 X.509

The IKE code supports the use of X.509 certificates for authentication. When the user configures a new peer for X.509 certificate authentication (via the "remote" CLI command), the user must specify the following items:

- A set of X.509 certificates that will be sent to the peer.
- An X.509 certificate for the peer.
- Use of `ASN.1 Distinguished Name` as the identification method.

Each certificate is sent to the peer inside a `CERT` payload, and the Identification information is sent in an ID payload.

To allow the received certificates from the peer to be validated, the user must use the "x509" CLI command to add certificates that can be used in the validation process. In addition to allowing the user to add certificates to the local X.509 certificate database, this command also allows the user to delete, list, and verify X.509 certificates.

3.12 IPSec API

IPSec Packet APIs

The IPSec toolkit provides a simple interface to the IP protocol stack. This interface is designed to be asynchronous. Network packets received by the IP protocol stack are sent to the IPSec toolkit. IPSec performs any required security operations and returns the packet back to the IP stack via a callback function. Every packet processed by the IP stack must be passed to the IPSec stack in order to ensure that the security policies are enforced.

Packets received by the IP stack that are addressed to the system are passed to the De-capsulation module. Packets received by the IP stack that are not addressed to the system and need to be routed are passed to the Encapsulation module. Packets originated on the system are also passed to the Encapsulation module.

On return from IPSec stack, the callback function is called and a Request structure is passed to the callback function. This structure contains, among other thing, the source packet, the destination packet, the policy applied and completion status.

The IPSec toolkit also includes synchronous APIs for easy integration with implementations that do not require asynchronous interface. This provides a simple interface for implementations that don't use hardware acceleration.

IPSec toolkit of InterNiche's Security Toolkit provides a simple interface to the IP protocol stack. This interface is designed to be asynchronous. Network packets received by the IP protocol stack are sent to the IPSec toolkit. IPSec performs any required security operations and returns the packet back to the IP stack via a callback function. Every packet processed by the IP stack must be passed to the IPSec stack in order to ensure that the security policies are enforced.

Packets received by the IP stack that are addressed to the system are passed to the Decapsulation module. Packets received by the IP stack that are not addressed to the system and need to be routed are passed to the Encapsulation module. Packets originated on the system are also passed to the Encapsulation module.

On return from IPSec stack, the callback function is called and a Request structure is passed to the callback function. This structure contains, among other thing, the source packet, the destination packet, the policy applied and completion status.

PacketDecapsulateSync

API Name

`PacketDecapsulateSync()`

Syntax

```
int PacketDecapsulateSync(Packet **pp, int domain, int *policyp)
```

Parameters

pp	Pointer to pointer to input packet (input), and pointer to pointer to output packet (output). The output packet can be one of the following: (1) encapsulated packet (when policy is SECURE), (2) input packet (when policy is BYPASS or DROP), or (3) NULL (an error occurred during processing)
domain	Domain on which this packet was received.
policyp	On successful completion, the policy applied to the packet. It is one of the following: (1) IPSEC_POLICY_BYPASS (IPSec processing was bypassed. The output packet is identical to the input packet. IP protocol processing must continue normally.), (2) IPSEC_POLICY_DROP (The packet must be dropped by the IP stack. The output packet is identical to the input packet.), (3) IPSEC_POLICY_SECURE_TUNNEL (The packet was secured in tunnel mode. The input packet has been replaced with the decapsulated packet. IP processing must be restarted on this packet for the encapsulated IP header.), or (4) IPSEC_POLICY_SECURE_TRANSPORT (The packet was secured in transport mode. The input packet has been replaced with the decapsulated packet.)

Description

This function is called to decapsulate a packet. The protocol field of the IP header is checked to see if the protocol is an IPSec protocol (AH or ESP). If the protocol field in the IP header of the packet indicates that it is an IPSec packet, then the packet is passed to the IPSec decapsulation module. If the protocol is not an IPSec protocol, then the packet is passed through the classification engine. If the policy for this packet indicates that the packet should have been an IPSec packet, then an error is returned.

Returns

This function returns 0 on success, non-zero error code on failure.

PacketEncapsulateSync

API Name

`PacketEncapsulateSync()`

Syntax

```
int PacketEncapsulateSync(Packet **pp, int domain, int *policyp)
```

Parameters

pp	Pointer to pointer to input packet (input), and pointer to pointer to output packet (output). The output packet can be one of the following: (1) encapsulated packet (when policy is SECURE), (2) input packet (when policy is BYPASS or DROP), or (3) NULL (an error occurred during processing)
domain	Domain on which this packet will be sent after IPSec.
policyp	On successful completion, the policy applied to the packet. It is one of the following: (1) IPSEC_POLICY_BYPASS (IPSec processing is bypassed. The output packet is identical to the input packet. IP protocol processing must continue normally.), (2) IPSEC_POLICY_DROP (The packet must be dropped by the IP stack. The output packet is identical to the input packet.), or (3a) IPSEC_POLICY_SECURE_TUNNEL / (3b) IPSEC_POLICY_SECURE_TRANSPORT (The packet has been secured. The input packet has been replaced with the encapsulated packet. The packet must be sent to the outbound network interface. IP must fragment the packet, if required, before sending it.)

Description

This function is called to encapsulate a packet. The packet is first passed through the classification engine. If the policy for this packet indicates that the packet must be secured, then it is passed to the IPSec engine and secured. Otherwise, it is returned back to the caller.

Returns

This function returns 0 on success, non-zero error code on failure.

PacketGetPolicy

API Name

`PacketGetPolicy()`

Syntax

```
int PacketGetPolicy(Packet *pkt, int domain, int *policyp, SP **spp)
```

Parameters

pp	Pointer to packet
domain	Domain identifier.
policyp	On successful completion, the policy that applies to the packet. It is one of the following: (1) IPSEC_POLICY_BYPASS (packet should bypass IPsec processing), (2) IPSEC_POLICY_DROP (packet should be dropped), (3) IPSEC_POLICY_SECURE_TUNNEL (packet should be secured in tunnel mode), or (4) IPSEC_POLICY_SECURE_TRANSPORT (packet should be secured in transport mode).
spp	Pointer to pointer to security policy structure. This pointer is filled in with the address of the matching security policy structure. (This parameter can be specified as NULL.)

Description

This function can be used to retrieve the policy that will be applied to the specified packet. The packet is classified, and if the classification is successful, the matching policy information is returned. The packet is not processed thru' IPsec. This function does not need to be explicitly called since the encapsulation process looks up the policy internally.

Returns

This function returns 0 on success, non-zero error code on failure.

Packet API

Management API

The IPSec module includes the Security Policy Database (SPD) and the Security Association Database (SAD). Management interface is needed for managing security policies and security associations. Management applications such as a command line interface (CLI) or a graphical user interface (GUI) as well as by the key management applications such as the Internet Key Exchange (IKE) protocol use the management interface to manage the policies and security associations.

IPSec toolkit of InterNiche's Security Toolkit provides two interfaces for managing the IPSec protocol stack. A set of management APIs and a message based interface. The message-based interface is a powerful interface that provides for complete control of the IPSec protocol. The management APIs provide a simple interface for use by management applications for configuring security policies and security associations. Although the message-based interface can be used by management applications, the APIs provide a simpler interface.

IPSecAdminAddBypassPolicy

API Name

`IPSecAdminAddBypassPolicy()`

Syntax

```
int IPSecAdminAddBypassPolicy(const char *srcid_str, const char *dstid_str,
Uchar protocol, Uint priority)
```

Parameters

srcid_str	Source Identity (traffic selector)
dstid_str	Destination Identity (traffic selector)
protocol	Upper layer (transport) protocol (e.g. TCP or UDP) (0 = any protocol)
priority	Policy priority

Description

This API is a wrapper function on top of IPSecMgmtAddPolicy() function and can be used instead of IPSecMgmtAddPolicy() for adding a policy to bypass IPsec processing on packets that match the specified traffic selectors. It takes NULL terminated ASCII strings for source and destination identities (traffic selectors).

Returns

This function returns 0 on success, non-zero error code on failure.

IPSecAdminAddDropPolicy

API Name

`IPSecAdminAddDropPolicy()`

Syntax

```
int IPSecAdminAddDropPolicy(const char *srcid_str, const char *dstid_str,
Uchar protocol, Uint priority)
```

Parameters

srcid_str	Source Identity (traffic selector)
dstid_str	Destination Identities (traffic selector)
protocol	Upper layer (transport) protocol (e.g. TCP or UDP) (0 = any protocol)
priority	Policy priority

Description

This API is a wrapper function on top of IPSecMgmtAddPolicy() function and can be used instead of IPSecMgmtAddPolicy() for adding a policy to drop packets that match the specified traffic selectors. It takes NULL terminated ASCII strings for source and destination identities (traffic selectors).

Returns

This function returns 0 on success, non-zero error code on failure.

IPSecAdminAddPolicy

API Name

`IPSecAdminAddPolicy()`

Syntax

```
int IPSecAdminAddPolicy(int policy, Uchar protocol, const char *srcid_str,
const char *dstid_str, const char *raddr_str, Uint32 flags, Uint priority,
Uint32 *spid)
```

Parameters

policy	Policy to apply (any one of IPSEC_POLICY_DROP, IPSEC_POLICY_BYPASS, IPSEC_POLICY_SECURE_TUNNEL, or IPSEC_POLICY_SECURE_TRANSPORT)
protocol	Upper layer (transport) protocol (e.g. TCP or UDP) (0 = any protocol)
srcid_str	Source Identity (traffic selector). Each identity is a NULL terminated string in any of the following formats: IP address, IP address/subnet mask, IP address/subnet bits, (start)IP address-(end)IP address, any, any6. Examples include "192.168.10.1", "192.168.10.0 /255.255.255.0", "192.168.10.0/24", "192.168.10.40-192.168.10.60", "192.168.10.1,1500", "192.168.10.0,1500/255.255.255.0", "192.168.10.0/24,1500", "192.168.10.40-192.168.10.60,1500", "3ffe:501:ffff::211:11ff:febe:7f61", "fe80::211:11ff: febe:7f61", "3ffe:501:ffff:2::/64", "any" (any IPv4 address), and "any6" (any IPv6 address).
dstid_str	Destination Identity (traffic selector)
raddr_str	Remote IPSec endpoint's IP address in standard notation. This can be NULL only if the policy is DROP or BYPASS.
flags	Policy flags (bitwise OR of any of the following: SP_PROTO_AH, SP_PROTO_ESP, SP_CHECK_REPLAY, SP_CHECK_ESP_PAD, SP_FULL_ECN, SP_DF_COPY, SP_DF_SET, SP_KEYNEG_MANUAL, SP_TX_DUMMY_PKT, SP_TX_TFC_PADDING, and SP_AUTH_SHA2_ICV_SHORT)
priority	Policy priority (any one of the following: SP_PRIORITY_HIGHEST, SP_PRIORITY_HIGH, SP_PRIORITY_MED_HIGH, SP_PRIORITY_MEDIUM, SP_PRIORITY_MED_LOW, SP_PRIORITY_LOW, or SP_PRIORITY_LOWEST)
spid	Pointer to integer that is used to store the policy identifier (output) (may be specified as NULL)

Description
This API is a wrapper function on top of IPSecMgmtAddPolicy function and can be used instead of IPSecMgmtAddPolicy for adding a policy. It takes NULL terminated ASCII strings for IP address and source and destination identities.

Returns
This function returns 0 on success, non-zero error code on failure.

IPSecMgmtAddPolicy

API Name

`IPSecMgmtAddPolicy()`

Syntax

```
int IPSecMgmtAddPolicy( int policy, const IPSecID *srcid, const IPSecID
*dstid, IPAddr *raddr, Uint32 flags, Uint priority, Uint32 *spid);
```

Parameters

policy	Policy to apply (any one of IPSEC_POLICY_DROP, IPSEC_POLICY_BYPASS, IPSEC_POLICY_SECURE_TUNNEL, or IPSEC_POLICY_SECURE_TRANSPORT).
srcid	Source Identity (traffic selector).
dstid	Destination Identity (traffic selector).
raddr	Remote IPSec endpoint's IP address. This can be NULL only if the policy is DROP or BYPASS.
flags	Policy flags (bitwise OR of any of the following: SP_PROTO_AH, SP_PROTO_ESP, SP_CHECK_REPLAY, SP_CHECK_ESP_PAD, SP_FULL_ECN, SP_DF_COPY, SP_DF_SET, SP_KEYNEG_MANUAL, SP_TX_DUMMY_PKT, SP_TX_TFC_PADDING, and SP_AUTH_SHA2_ICV_SHORT).
priority	Policy priority (any one of the following: SP_PRIORITY_HIGHEST, SP_PRIORITY_HIGH, SP_PRIORITY_MED_HIGH, SP_PRIORITY_MEDIUM, SP_PRIORITY_MED_LOW, SP_PRIORITY_LOW, or SP_PRIORITY_LOWEST)
spid	Pointer to integer that is used to store the policy identifier (output) (may be specified as NULL).

Description

This API is used for adding a policy into the IPsec security policy database (SPD). This API can be used either with manual keying (static security association (SA)) or with automated keying (IKE).

Returns

This function returns 0 on success, non-zero error code on failure.

IPSecMgmtAddSA

API Name

`IPSecMgmtAddSA()`

Syntax

```
int IPSecMgmtAddSA (const Uint32 spid, Uchar encr_alg, Uchar encr_keylen,
const Uchar *encr_i_key, const Uchar *encr_o_key, Uchar auth_alg, Uchar
auth_keylen, const Uchar *auth_i_key, const Uchar *auth_o_key, Uint32
esp_i_spi, Uint32 esp_o_spi, Uint32 ah_i_spi, Uint32 ah_o_spi);
```

Parameters

spid	Policy ID
encr_alg	Encryption algorithm (ALG_ESP_*)
encr_keylen	Encryption key length (bytes)
encr_i_key	Inbound encryption key
encr_o_key	Outbound encryption key
auth_alg	Authentication algorithm (ALG_ESP_HMAC_* or ALG_AH_*)
auth_keylen	Authentication key length (bytes)
auth_i_key	Inbound authentication key
auth_o_key	Outbound authentication key
esp_i_spi	ESP inbound SPI (host byte order)
esp_o_spi	ESP outbound SPI (host byte order)
ah_i_spi	AH inbound SPI (host byte order)
ah_o_spi	AH outbound SPI (host byte order)

Description

This is used to add a security association into SAD for manual keying of IPSec. It is only used if IKE is not used for key negotiation. The policy must be added before an SA can be added. If both AH and ESP protocols are used for an SA, the inbound SPIs must be same for both protocols.

Returns

This function returns 0 on success, non-zero error code on failure.

3.13 IPSec CLI

The IPSec and IKE packages provide several CLI utilities to assist in development of your target application. Inclusion of these in your target application is dependent on whether `INCLUDE_CLI` is defined in `ipport.h`.

ipsec addsa

Command Name

`ipsec addsa - add IPSec security association`

Syntax

```
ipsec addsa {-a -b -c} {-e -f -g} {-m -p -r -s -d} {-w -x -y -z}
```

Parameters

`-a`	authentication protocol.
`-b`	incoming authentication key>
`-c`	outgoing authentication key.
`-e`	encryption protocol.
`-f`	incoming encryption key.
`-g`	outgoing encryption key.
`-m`	IPsec mode (transport or tunnel).
`-p`	protocol number.
`-r`	remote endpoint.
`-s`	source id string.
`-d`	destination id string.
`-w`	incoming ESP SPI.
`-x`	outgoing ESP SPI.
`-y`	incoming AH SPI.
`-z`	outgoing AH SPI.

Description

This command adds an IPsec security association with the specified parameters.

Notes/Status

Here are example CLI commands that illustrate the configuration of ESP-only, AH-only, and ESP+AH static security associations. Each example contains two CLI commands, one for each of the two InterNiche TCP/IP stacks being used to establish a secure connection. The AES and 3DES privacy algorithms can be specified via "AES" and "3DES" respectively. The SHA2 family of authentication algorithms are identified via the following names: "SHA256", "SHA384", and "SHA512".

- ESP-only, transport mode, encryption (DES-CBC) and authentication (SHA1), all protocols ("any"), IPv4

```
ipsec addsa -a SHA -b 0000000000000000000000000000000bbbbbbbb
    -c 0000000000000000000000000000000aaaaaaaa
    -e DES -f 00000000ffffffff -g 00000000eeeeeeee -m transport -p 0
    -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -w 0xeeeeeeee -x 0xffffffff

ipsec addsa -a SHA -b 0000000000000000000000000000000aaaaaaaa
    -c 0000000000000000000000000000000bbbbbbbb
    -e DES -f 00000000eeeeeeee -g 00000000ffffffff -m transport -p 0
    -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -w 0xffffffff -x 0xeeeeeeee
```

- ESP-only, tunnel mode, encryption (DES-CBC) and authentication (SHA1), all protocols ("any"), IPv4

```
ipsec addsa -a SHA -b 0000000000000000000000000000000bbbbbbbb
    -c 0000000000000000000000000000000aaaaaaaa
    -e DES -f 00000000ffffffff -g 00000000eeeeeeee -m tunnel -p 0
    -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -w 0xeeeeeeee -x 0xffffffff

ipsec addsa -a SHA -b 0000000000000000000000000000000aaaaaaaa
    -c 0000000000000000000000000000000bbbbbbbb
    -e DES -f 00000000eeeeeeee -g 00000000ffffffff -m tunnel -p 0
    -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -w 0xffffffff -x 0xeeeeeeee
```

- ESP-only, transport mode, encryption (DES-CBC) and authentication (MD5), all protocols ("any"), IPv4

```
ipsec addsa -a MD5 -b 000000000000000000000000bbbbbbbb
    -c 000000000000000000000000aaaaaaaa -e DES
    -f 00000000ffffffff -g 00000000eeeeeeee -m transport -p 0
    -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -w 0xeeeeeeee -x 0xffffffff

ipsec addsa -a MD5 -b 000000000000000000000000aaaaaaaa
    -c 000000000000000000000000bbbbbbbb -e DES
    -f 00000000eeeeeeee -g 00000000ffffffff -m transport -p 0
    -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -x 0xeeeeeeee -w 0xffffffff
```

- ESP-only, tunnel mode, encryption (DES-CBC) and authentication (MD5), all protocols ("any"), IPv4

```
ipsec addsa -a MD5 -b 000000000000000000000000bbbbbbbb
    -c 000000000000000000000000aaaaaaaa -e DES
    -f 00000000ffffffff -g 00000000eeeeeeee -m tunnel -p 0
    -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -w 0xeeeeeeee -x 0xffffffff

ipsec addsa -a MD5 -b 000000000000000000000000aaaaaaaa
    -c 000000000000000000000000bbbbbbbb -e DES
    -f 00000000eeeeeeee -g 00000000ffffffff -m tunnel -p 0
    -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -w 0xffffffff -x 0xeeeeeeee
```

- AH-only, transport mode, authentication (MD5), all protocols ("any"), IPv4

```
ipsec addsa -a MD5 -b 000000000000000000000000bbbbbbbb
    -c 000000000000000000000000aaaaaaaa -m transport
    -p 0 -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -y 0xeeeeeeee -z 0xffffffff

ipsec addsa -a MD5 -b 000000000000000000000000aaaaaaaa
    -c 000000000000000000000000bbbbbbbb -m transport
    -p 0 -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -y 0xffffffff -z 0xeeeeeeee
```

- AH-only, tunnel mode, authentication (MD5), all protocols ("any"), IPv4

```
ipsec addsa -a MD5 -b 0000000000000000000000bbbbbbbb
    -c 0000000000000000000000aaaaaaaa -m tunnel
    -p 0 -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -y 0xeeeeeeee -z 0xffffffff

ipsec addsa -a MD5 -b 0000000000000000000000aaaaaaaa
    -c 0000000000000000000000bbbbbbbb -m tunnel
    -p 0 -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
   -y 0xffffffff -z 0xeeeeeeee
```

- AH-only, transport mode, authentication (SHA1), all protocols ("any"), IPv4

```
ipsec addsa -a SHA -b 0000000000000000000000000000000bbbbbbbb
    -c 0000000000000000000000000000000aaaaaaaa
    -m transport -p 0 -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -y 0xeeeeeeee -z 0xffffffff

ipsec addsa -a SHA -b 0000000000000000000000000000000aaaaaaaa
    -c 0000000000000000000000000000000bbbbbbbb
    -m transport -p 0 -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -z 0xeeeeeeee -y 0xffffffff
```

- AH-only, tunnel mode, authentication (SHA1), all protocols ("any"), IPv4

```
ipsec addsa -a SHA -b 0000000000000000000000000000000bbbbbbbb
    -c 0000000000000000000000000000000aaaaaaaa
    -m tunnel -p 0 -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -y 0xeeeeeeee -z 0xffffffff

ipsec addsa -a SHA -b 0000000000000000000000000000000aaaaaaaa
    -c 0000000000000000000000000000000bbbbbbbb
    -m tunnel -p 0 -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -z 0xeeeeeeee -y 0xffffffff
```

- ESP+AH, transport mode, encryption (DES-CBC) (with ESP), authentication (MD5) (with AH), all protocols ("any"), IPv4

```
ipsec addsa -a MD5 -b 0000000000000000000000000bbbbbbbb
    -c 0000000000000000000000000aaaaaaaa -e DES
    -f 00000000ffffffff -g 00000000eeeeeeee -m transport -p 0
    -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -w 0xeeeeeeee -x 0xffffffff -y 0xeeeeeeee -z 0xffffffff

ipsec addsa -a MD5 -b 0000000000000000000000000aaaaaaaa
    -c 0000000000000000000000000bbbbbbbb -e DES
    -f 00000000eeeeeeee -g 00000000ffffffff -m transport -p 0
    -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -w 0xffffffff -x 0xeeeeeeee -y 0xffffffff -z 0xeeeeeeee
```

- ESP+AH, tunnel mode, encryption (DES-CBC) (with ESP), authentication (MD5) (with AH), all protocols ("any"), IPv4

```
ipsec addsa -a MD5 -b 0000000000000000000000000bbbbbbbb
    -c 0000000000000000000000000aaaaaaaa -e DES
    -f 00000000ffffffff -g 00000000eeeeeeee -m tunnel -p 0
    -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -w 0xeeeeeeee -x 0xffffffff -y 0xeeeeeeee -z 0xffffffff

ipsec addsa -a MD5 -b 0000000000000000000000000aaaaaaaa
    -c 0000000000000000000000000bbbbbbbb -e DES
    -f 00000000eeeeeeee -g 00000000ffffffff -m tunnel -p 0
    -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -w 0xffffffff -x 0xeeeeeeee -y 0xffffffff -z 0xeeeeeeee
```

- ESP+AH, transport mode, encryption (DES-CBC) (with ESP), authentication (SHA1) (with AH), all protocols ("any"), IPv4

```
ipsec addsa -a SHA -b 0000000000000000000000000000000bbbbbbbb
    -c 0000000000000000000000000000000aaaaaaaa
    -e DES -f 00000000ffffffff -g 00000000eeeeeeee -m transport -p 0
    -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -w 0xeeeeeeee -x 0xffffffff -y 0xeeeeeeee -z 0xffffffff

ipsec addsa -a SHA -b 0000000000000000000000000000000aaaaaaaa
    -c 0000000000000000000000000000000bbbbbbbb
    -e DES -f 00000000eeeeeeee -g 00000000ffffffff -m transport -p 0
    -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -w 0xffffffff -x 0xeeeeeeee -y 0xffffffff -z 0xeeeeeeee
```

- ESP+AH, tunnel mode, encryption (DES-CBC) (with ESP), authentication (SHA1) (with AH), all protocols ("any"), IPv4

```
ipsec addsa -a SHA -b 0000000000000000000000000000000bbbbbbbb
    -c 0000000000000000000000000000000aaaaaaaa
    -e DES -f 00000000ffffffff -g 00000000eeeeeeee -m tunnel -p 0
    -r 10.0.0.112 -s 10.0.0.114 -d 10.0.0.112
    -w 0xeeeeeeee -x 0xffffffff -y 0xeeeeeeee -z 0xffffffff

ipsec addsa -a SHA -b 0000000000000000000000000000000aaaaaaaa
    -c 0000000000000000000000000000000bbbbbbbb
    -e DES -f 00000000eeeeeeee -g 00000000ffffffff -m tunnel -p 0
    -r 10.0.0.114 -s 10.0.0.112 -d 10.0.0.114
    -w 0xffffffff -x 0xeeeeeeee -y 0xffffffff -z 0xeeeeeeee
```

Location

This command is provided by the `IPsec` module when `IPSEC` is defined.

ipsec delsa

Command Name

`ipsec delsa - delete IPSec security association`

Syntax

```
ipsec delsa -d -p -s
```

Parameters

`-d`	destination id string.
`-p`	protocol number.
`-s`	source id string.

Description

This command deletes a security association with the specified source id string, destination id string, and protocol number.

Location

This command is provided by the `IPsec` module when `IPsec` is defined.

ipsec flush

Command Name

`ipsec flush - flush IPsec security association and/or policy database`

Syntax

```
ipsec flush -a -c -p
```

Parameters

`-a`	flush IPsec security association and security policy databases.
`-c`	flush IPsec security association database.
`-p`	flush IPsec security policy database.

Description

This command flushes IPsec security association and/or security policy databases.

Location

This command is provided by the `IPsec` module when `IPsec` is defined.

ipsec netstat

Command Name

`ipsec netstat - display IPsec security association and/or policy database`

Syntax

```
ipsec netstat -c -p
```

Parameters

`-c`	display information about IPsec security associations.
`-p`	display information about IPsec security policies.

Description

This command displays information about IPsec security associations and/or security policies.

Notes/Status

- When no options are specified, this command displays information about IPsec security associations and security policies.

Location

This command is provided by the `IPsec` module when `IPsec` is defined.

3.14 IPSec Messaging Interface

Most implementations may not need to use this interface. The management APIs provides a simpler way to manage IPSec. As such this chapter may be ignored by such implementers.

Inclusion of these features in your build is controlled by the following line in `ipport.h`:

```
#define IPSEC     1 /* IPSec */
```

The IPSec component of InterNiche's Security Toolkit provides a message-based interface for managing the IPSec protocol. IPSec module includes the Security Policy Database (SPD) and the Security Association Database (SAD). Management interface is provided for managing security policies and security associations. This interface is used by management applications such as a command line interface (CLI) or a graphical user interface (GUI) as well as by the key management applications such as the Internet Key Exchange (IKE) protocol.

Messages Overview

The IPSec management interface is based on a message-based protocol. All messages are of request-reply type. Each request has a corresponding reply to ensure reliability. Most request messages originate from the management application; however, asynchronous event notification can originate from the IPSec protocol module.

Each message consists of a message header followed by a message specific data. Messages are usually less then 1024 bytes long so that contiguous buffer space can be allocated in the IPSec protocol for easy manipulation of the data.

To ensure reliability, each request has a corresponding reply.

The message originator and target are assumed to be on the same system and there is no attempt made to account for possible byte-order mismatch if the originator and target are not on the same system. This only affects the IPSec management message header and not the data in the message.

Message Header

The message header is eight octets long. The header is the same on all requests and reply messages. The fields of the message header are shown below:

Field	Description
type	The message code.
src	The identity of the originator of the message (e.g., IKE protocol).
dst	The target of the message (e.g. the IPSec protocol).
reserved	Currently not defined and must be set to zero.
sequence	The number used to identify the message. It is set by the originator in the request message and is copied from the request message to the reply message by the target.
length	The length of the message data not including the message header.
status	is set to zero in the request and set to the completion status in the reply.
padding	is unused by the software and exists to avoid alignment issues associated with some processor architectures.

Message Types

There are two types of messages: message requests and message events. Message requests are generated by a management application (such as IKE) and sent to IPSec. Message events are generated by IPSec and sent to all registered applications (such as IKE).

The message type field identifies the type of the message. The following types are currently defined:

Type	Originator	Target	Description
APP_REGISTER	IKE	IPSec	Register for event notification
CLONE_POLICY	IKE	IPSec	Clone a policy
INITIAL_CONTACT	IKE	IPSec	Initial Contact received from peer
DELETE_SPIs	IKE	IPSec	Delete SPI received from peer
GET_SPI	IKE	IPSec	Get inbound SPI
ACQUIRE_FAIL	IKE	IPSec	Acquire SA failure notification
ADD_SA	IKE/ADMIN	IPSec	Add an IPSec SA
GET_POLICY	IKE/ADMIN	IPSec	Get specified policy from SPD
GET_POLICIES	IKE/ADMIN	IPSec	Sequentially get policies from SPD
LOOKUP_POLICY	IKE/ADMIN	IPSec	Lookup a policy in SPD
SAFLUSH	IKE/ADMIN	IPSec	Delete all SAs from SAD
ADD_POLICY	ADMIN	IPSec	Add a policy in the SPD
DEL_POLICY	ADMIN	IPSec	Delete a policy from the SPD
GET_SA	ADMIN	IPSec	Get SA for specified policy
DEL_SA	ADMIN	IPSec	Delete SA for specified policy
FLUSH_SPD	ADMIN	IPSec	Delete all policies from SPD
ACQUIRE	IPSec	IKE	Negotiate SAs with peer
SA_EXPIRED	IPSec	IKE	SA expired notification
SA_DELETED	IPSec	IKE	SA deleted notification
POLICY_ADDED	IPSec	IKE	Policy added notification
POLICY_DELETED	IPSec	IKE	Policy deleted notification

Message Description

All messages consist of a request and a response. Each message (either request or response) has a message header. There may be additional data associated either with the request or with the response depending on the message type.

MGMT_TYPE_APP_REGISTER

This message is used to register the application with IPSec so it can receive events from IPSec.

There is no data associated with the request message.

There is no data associated with the response message.

MGMT_TYPE_CLONE_POLICY

This message is used by the application to tell IPSec to clone a policy. This can be used by IKE to create a cloned policy in the SPD based on the traffic selectors that it received as an IKE responder from its peer. This is an optional capability that most implementations of IKE do not currently support.

The data for the request message consists of the policy ID structure MgmtPolicyID.

There is no data associated with the response message.

This message is currently not supported either by IKE or IPSec

MGMT_TYPE_INITIAL_CONTACT

This message sent by IKE to IPSec when it receives an Initial Contact message from its peer. This will inform IPSec to clear any security associations that IPSec may have in its SAD.

The data for the request message consists of the structure MgmtInitialContact.

There is no data associated with the response message.

MGMT_TYPE_DELETE_SPIs

This message is used by the application to tell IPSec to delete IPSec sessions associated with the specified SPIs. Several sessions can be deleted with a single request message.

The data for the request message consists of the structure MgmtDeleteSPI.

The spi_size in the message must be 4 for IPSec SPIs. Since IPSec treats the combination of AH with ESP as logically a single protocol and always assigns the same SPI to both AH and ESP when they are used together, they must be deleted together. Hence the proto field must be set to IPSEC_PROTO_ANY when deleting SPIs and only one SPI must be specified when using the ESP and AH together.

There is no data associated with the response message.

MGMT_TYPE_ACQUIRE_FAIL

This message is sent by an application IPSec when the application fails to establish SA with a peer in response to a ACQUIRE event from IPSec. Note that this message is an independently generated request message and not a response to the ACQURIE event message.

The data for the request message consists of the structure MgmtPolicyID.

There is no data associated with the response message.

MGMT_TYPE_GET_SPI

This message is used by the application to obtain an SPI from IPSec for a given policy. Either AH or ESP or both SPIs can be obtained with a single request.

The data for the request message consists of the structure MgmtGetSPI.

The data for the response message consists of the structure MgmtGetSPI.

3.15 IPSec Error Codes

Error handling in the IPSec toolkit of InterNiche's Security Toolkit is provided by returning a descriptive error code. Several error codes are provided to identify the error unambiguously. Since IPSec is asynchronous, errors are usually returned in the Request structure, which is passed as a parameter to the IPSec completion callback function.

The following is the list of error codes and a brief description of each error.

Error Code	Description
XE_AH_DIGEST_MISMATCH	AH Digest mismatch
XE_BAD_ADDR_RANGE	Bad IP Address Range
XE_BAD_AH_SPI	Bad AH SPI
XE_BAD_ESP_SPI	Bad ESP SPI
XE_BAD_ID_TYPE	Bad ID type
XE_BAD_PAD_DATA	Bad Pad Data
XE_BAD_PARAM	Bad parameter
XE_BAD_PORT_NUMBER	Bad TCP or UDP port number
XE_BAD_SEQUENCE_NUMBER	Replay failure - Bad AH or ESP sequence number
XE_BAD_SESS_ERR	Bad IPsec session status
XE_DST_OUT_OF_DATA	Ran out of data on destination packet
XE_DST_OUT_OF_SPACE	Ran out of space in destination packet
XE_ESP_DIGEST_MISMATCH	ESP Digest mismatch
XE_EXIST	An identical entry already exists
XE_EXPECTED_AH_PROTO	Expected next_proto to be AH
XE_EXPECTED_ESP_PROTO	Expected next_proto to be ESP
XE_INBOUND_EXPECTED_SECURE_PACKET	Received an unsecured packet when policy specifies SECURE
XE_INBOUND_POLICY_FAILURE	Inbound Policy Mismatch
XE_INBOUND_UNEXPECTED_SECURED_PACKET	Received an IPSec secured packet when policy specifies DROP or BYPASS
XE_KEYNEG_INPROGRESS	Key negotiation is in progress

Error Code	Description
XE_MTU_EXCEEDED	Packet larger than MTU
XE_NOMEM	Out of memory
XE_NO_DATA	Not enough data
XE_OUT_OF_PACKETS	Unable to allocate a Packet
XE_OUT_OF_POLICIES	Unable to allocate a Policy
XE_OUT_OF_REQS	Unable to allocate a Request
XE_OUT_OF_SESSIONS	Unable to allocate a Session
XE_POLICY_NOT_FOUND	Policy Not Found
XE_SA_NOT_FOUND	Security Association not found
XE_SESS_UNAVL_FOR_DUMMY_TX	Secure session can't be used to transmit a dummy packet
XE_SRC_OUT_OF_DATA	Ran out of data on source packet
XE_UNKNOWN_ALGORITHM	Unknown Algorithm
XE_UNKNOWN_IP_PROTOCOL	Unknown IP protocol

4 IKE

The Internet Key Exchange (IKE) protocol is used for key agreement and key management within IPSec. It provides a scalable method of generating security associations for IPSec protocols. The protocol includes system authentication, key agreement and security association generation. This protocol is specified by the IETF standard RFC 2409. Several other RFCs specify related standards.

4.1 IKE Quick Start

Internet Key Exchange, IKE, provides key management for IPSec. IPSec provides security, encryption and authentication, for IP communications. IPSec could be used without IKE either with another key exchange protocol or by manually providing keys to both endpoints. Manual provisioning of IPSec could be difficult to manage. IKE simplifies the key management.

IKE comes in two version, IKEv1 and IKEv2; InterNiche's IKE supports both. The two versions are incompatible with each other. A node initiating the key exchange using IKEv1 will need the receiving node to use IKEv1 as well. A node initiating the exchange using IKEv2 will require the other node to use IKEv2 in response. InterNiche's implementation of IKE can be configured to initiate one of IKEv1 or IKEv2 and it can be configured to accept either a key exchange from either version.

Because IKE works with IPSec there are two primary configuration commands, the "`ipsec policy`" command and the "`ike remote`" command. IPSec's `policy` command describes the parameters that IPSec will need to secure communications with a remote node. There can be multiple policies for a remote node, for example, one policy for UDP traffic and another policy for TCP traffic. IKE's "`remote`" command describes the parameters that IKE needs to exchange keys with the remote node. IPSec and IKE support both IPv4 and IPv6. A separate "`ipsec policy`" command and "`ike remote`" command is needed for each. Therefore, one may have two "`ike remote`" commands for a single remote node: one for IPv4 and another for IPv6.

Both the `policy` command and the `remote` command require a "`-n <NAME>`" parameter. This applies a name to the IPSec `policy` or IKE `remote` command that is used in the configuration and CLI of IPSec and IKE. The name has no significance outside of the CLI, in particular, it does not need to reflect the node's host name or DNS name. Information and statistics can be displayed using the name for example "`ike remote -v -n <NAME>`"; or the configuration information can be removed, "`ike remote -x <NAME>`". The "`-g <NAME>`" parameter in IPSec's `policy` command refers to the name found in one of IKE's `remote` commands.

IKE must negotiate and exchange key information securely. IKE uses encryption and authentication algorithms. The algorithms used by IKE during the key exchange could be different from the algorithms chosen during the negotiation to be used by IPSec. The "ipsec policy" command specifies the algorithms that IPSec will support while the "ike remote" command specifies the algorithms that IKE will support to carry out the negotiation. These algorithms could be different in each case and therefore are specified in each command.

IPv6 requires a brief time to establish the addresses for each interface, listen for router advertisements, perform duplicate address checks, etc. After a brief startup delay the NicheStack IKE module will read the "ike_rc" file and execute the commands found. This is best place for the "ipsec policy" and "ike remote" commands.

IKEv1 and IKEv2 are different protocols with some different options. Therefore, when configuring IKE with the "ike remote" command a parameter is required to specify which version of the protocol the command is configuring; the "-z <VERSION>" specifies whether the command line applies to IKEv1 or IKEv2. If both are being used to accept either an IKEv1 or IKEv2 exchange initiated by the remote host then two command lines are required; one that contains "-z 1" and one line that contains "-z 2". The two parameters together "-z 2 -w" indicate that IKEv2 is being configured and IKEv2 will be used when initiating a key exchange. The "-u" parameter indicates that the version being configured can be used to respond to a key exchange initiated by the remote. It would be common to use "-z 2 -w -u" together to indicate that IKEv2 will be used to initiate the key exchange and will be accepted when a remote host initiates the key exchange.

During key exchange IKE can use either a pre-shared key or a certificate to authenticate the the remote node. In the reference port the pre-shared key is stored in a file whose name is provided in the "ike remote" "-h <FILENAME>" parameter. The porting engineer should take appropriate steps to insure that this file (if used) is secure so that the contents remain secure.

In the reference port both the "ike_rc" file and the "secret_file.txt" are in the same directory as the executable.

IPSec will secure traffic that matches a given traffic selector. Traffic selectors specify the source and destination by ip address and port number as well as the protocol type, such as UDP or TCP. Each "ipsec policy" command defines a traffic selector. Each "ipsec policy" command contains a "-g <NAME>" parameter which points to a corresponding "ike remote" command which has that same <NAME> in the "-n <NAME>" parameter of the remote command. That "ike remote" command defines the IKE parameters for the key negotiation. For example:

```
ipsec policy -n Policy3 -g mongo -y -l 4 -p udp -m transport -t ESP -a HMAC-MD5 -e NULL_ENC -r
10.0.0.76 -s 10.0.0.140 -d 10.0.0.76

ike remote -n mongo -z 2 -w -u -a HMAC-MD5 -e 3DES-CBC -d MODP1024 -p HMAC-MD5 -m PresharedKey -
h secret_file.txt -l 86400 -i 1 -r 10.0.0.76 -f ipv4:10.0.0.76 -g ipv4:10.0.0.140
```

Multiple "ipsec policy" commands for different traffic types can refer to the same "ike remote" command. For example:

```
ipsec policy -n Policy2 -g mongo -y -l 4 -p tcp -m transport -t ESP -a HMAC-MD5 -e NULL_ENC -r
10.0.0.76 -s 10.0.0.140 -d 10.0.0.76
```

In this example both Policy2 (for tcp traffic) and Policy3 (for udp traffic) refer to the "ike remote" command for "mongo".

The man pages for "ipsec policy" and "ike remote" provide details for the command line parameters for each command.

4.2 IKE Architecture

The IKE module uses standard POSIX library calls for the following operating system specific functions:

- Standard I/O - printf function (desirable but not required) for IKE packet decoding and debugging.
- Socket APIs -- used for communication with the IKE Peer.
- syslog - (Optional) message logs for debugging purposes
- File I/O - (Optional) message logs for debugging purposes
- Time - Current system time
- InterNiche Crypto Library or OpenSSL - Public key and symmetric key cryptography. OpenSSL includes support for X509 certificates and RSA/DSA signatures as well as built-in support for hardware acceleration.

4.3 IKE Cryptographic Library

IKE requires symmetric key and public key cryptography. The symmetric key algorithms include the encryption (e.g. DES, 3DES or AES) and digest algorithms (e.g. SHA-1, MD5). The required public key algorithm is Diffie-Hellman key agreement algorithm. In addition, the RSA algorithm will be required if you plan to support X.509 certificate-based authentication.

The cryptographic library included with IKE provides the following algorithms:

- DES, Triple DES, and AES (128-bits, 192-bits, and 256-bits)
- MD5, SHA-1, SHA-2 (256-bits, 384-bits, and 512-bits)
- Diffie-Hellman

4.4 IKE Startup

The function `tk_ikev2()` (`ikev2/ikev2_mod.c`) contains the code for the IKE task. This function invokes code to perform the required initialization, and then processes various events such as messages from peer IKE entities, messages from the IPsec module, etc.

`ikev2_init_and_cfg_cmn` is called from TK_ENTRY(). It installs bypass policies on the IPv4 address of each interface for UDP traffic to the IKE port. If IPv6 is configured it will also install a bypass policy for Neighbor Discovery traffic. After waiting 1 minute for IPv6 addresses it will install bypass policies for UDP traffic to the IKE port of all link-local and global IPv6 addresses.

It also installs one low priority bypass policy for all IPv4 traffic and one for all IPv6 traffic. These policies can be useful while testing because it is often helpful to allow some traffic to/from the device that is not handled by ipsec. However, these policies should not be part of a production build because it obviously bypasses security. These two low priority bypass policies are included when the following line is enabled in `ipport.h` .

```
#define IKE_ADD_TEST_BYPASS_POLICIES  1
```

`rc_main()` initializes buffer pools for IKE and gets all ip addresses. `isakmp_init` opens sockets to listen for IKE messages on these addresses.

Next IKE policy and remote peer information needs to be added. This is most easily done using the `ike_rc` file, which can contain "`ipsec policy`" and "`ike remote`" commands. Alternatively, the CLI commands could be used directly from the console, or the porting engineer could use the API's `ikev2_AddPolicy` and `ikev2_CreateRemote`.

The "policy" command in ipsec_policy/ipsec_nt.c adds information for IPSec and IKE. This information can be seen using the command "`ipsec policy -v [-n <name>]`". The "IKE netstat -p" command shows part of this information. Some of this information is used by IKE during key negotiation. For example it uses the lists of authentication and encryption algorithms. Part of the information is used by ipsec, such as, the src, dst, and protocol type to classify packets and look for a matching policy.

The "remote" command in ikev2_Remote/ikev2_nt.c adds information about an IKE remote peer. When a packet is sent or received that matches a policy to secure the packet but does not have a current SA then IKE will be invoked to establish an SA. IKE will use the associated remote information to connect with the remote peer, negotiate, and establish a secure IKE SA. This secure IKE connection will then be used to negotiate and establish a second secure SA for the specified packets.

The information in the "remote" command is used to establish the IKE SA while the information in the "policy" command is used for SAs for the traffic defined in the policy (src, dst, protocol). The algorithms used for the IKE SA (as specified in the "remote" command) may be different from the algorithms used for the IPSec SA (as specified in the "policy" command).

The "policy" specifies which remote peer to use through the "-g" parameter which must match the name of a remote as specified in the "-n" parameter of the corresponding "remote" command. More than one policy may specify the same remote peer. The commands can be entered in any order. A runtime error will occur if a policy is used which specifies a remote that has not been configured.

4.5 IKE Debugging and Packet Decoder

The IKE toolkit includes a packet decoder utility for decoding IKEv1 packets and displaying all the packet contents in an easy to understand format with both textual message and hexadecimal data. This utility can decode IKEv1 packets even when they are encrypted.

When the IKE toolkit is built with `IKE_DEBUG` and `IKE_DEBUG_TRACE` enabled various debugging messages will be printed including a packet trace which will decode and display both IKEv1 and IKEv2 packets.

```
10-10-2002 10:19:02.353 84 bytes; remote 127.0.0.1 [5432]; Received from Remote
  0000  f1 40 8d c4 fd 62 1c c4   Initiator Cookie
  0008  00 00 00 00 00 00 00 00   Responder Cookie
  0010  01                        Next Payload - Security Association
  0011  10                        Version: 1.0
  0012  02                        Exchange Type - Identity Protection (Main Mode)
  0013  00                        Flags -
  0014  00 00 00 00               Message ID
  0018  00 00 00 54               Length
  001c  00                        Next Payload - NONE
  001d  00                        Reserved
  001e  00 38                     Payload Length - 56
  0020  00 00 00 01               DOI - IPSEC
  0024  00 00 00 01               Situation - SIT IDENTITY ONLY
  0028  00                        Next Payload - NONE
  0029  00                           Reserved
  002a  00 2c                        Payload Length - 44
  002c  01                           Proposal Number - 1
  002d  01                           Protocol ID - PROTO_ISAKMP
  002e  00                           SPI Size - 0
  002f  01                           Number of Transforms - 1
  0030  00                              Next Payload - NONE
  0031  00                              Reserved
  0032  00 24                           Payload Length - 36
  0034  01                              Transform Number - 1
  0035  01                              Transform ID - KEY_IKE
  0036  00 00                           RESERVED
  0038  80 0b 00 01                     ATTR: Life Type      - seconds
  003c  00 0c 00 04 00 01 51 80         ATTR: Life Duration  - 86400
  0044  80 01 00 05                     ATTR: Encryption     - 3DES-CBC
  0048  80 03 00 01                     ATTR: Authentication - Pre-shared Key
  004c  80 02 00 02                     ATTR: Hash           - SHA
  0050  80 04 00 02                     ATTR: Group          - 2
```

IKE Packet Decoder Sample Output

4.6 Configuration

IKE and IPSec can be configured using either APIs or CLI commands. The `ikev2_AddPolicy` API (or "`ipsec policy`" CLI command) defines the parameters needed by IPSec and IKE for secure data transfer. IKE negotiates with the remote peer to find a set of parameters (such as encryption algorithms) that each side is willing to support during data transfer. There may be multiple policies for each associated "`ike remote`" command/configuration and there must be an "`ike remote`" configuration for each IKE peer.

The `ikev2_CreateRemote` API (or "`ike remote`" CLI command) provides the parameters needed by IKE to establish communication with a remote peer. One command is needed for each remote peer.

4.7 IKE Administrative APIs

Policy

ikev2_AddPolicy

Adds IPSec and IKE information for specific policies, algorithm choices, etc

ikev2_DeletePolicy

ikev2_DeleteAllPolicies

Removes IPSec and IKE policy info

IkeAdminPrintLocalConf

Prints IKE information about a policy

Remote

ikev2_CreateRemote

Creates IKE information about remote peer; Calls ikeCreateKmp which uses ikev1_default_values or ikev2_default_values except as overridden by passed parameters

ikev2DeleteAllRemotes

ikev2DeleteRemote

Deletes database information about remote peers

ikev2_shutdown

Deletes Security Associations, SAs

IkeAdminPrintRemoteConf

Prints IKE information about remote peers

4.8 IKE CLI

The IPSec and IKE packages provide several CLI utilities to assist in development of your target application. Inclusion of these in your target application is dependent on whether `INCLUDE_CLI` is defined in the file `ipport.h`.

ike netstat

Command Name

`ike netstat - display IKE sessions and remote configuration database`

Syntax

```
ike netstat [-p <1 | 2 | 3>] [-r [-n <remote name>]]
```

Parameters

`-p`	display information about IKEv1 Phase 1 or 2 session(s) or -p 3 implies IKEv2 SA.
`-r`	display contents of IKE remote configuration database.
`-n`	Name in remote configuration database.

Description

This command displays information about Security Assocications, SA(s). IKEv1 Phase 1 session(s), IKEv1 Phase 2 session(s), IKEv2 SA(s), and remote configuration database.

Notes/Status

- When no options are specified, this function displays information about IKEv1 Phase 1 session (s), IKEv1 Phase 2 session(s), IKEv2 SA(s), and remote configuration database. "-p 3" implies IKEv2 SA(s).
- The -n option is only intended for use with the -r option.

Location

This command is provided by the `IKEv2` module when `IKEv2` is defined.

ike flush

Command Name

`ike flush - flush IKE SAs and remote configuration database`

Syntax

```
ike flush {-p <1 | 2 | 3>} {-r [-n <name of remote peer>]}
```

Parameters

`-p`	flush all IKEv1 Phase 1 (-p 1) or Phase 2 (-p2) SAs or all IKEv2 SAs (-p 3).
`-r`	flush contents of IKE remote configuration database.
`-n`	name of remote peer as it appears in the remote peer database

Description

This command flushes IKEv1 Phase 1 and Phase 2 SAs and IKEv2 SAs, and remote configuration database.

Notes/Status

- If no options are given then all SAs for both IKEv1 and IKEv2 are deleted and all remote peer information is deleted.
- The -n option is only intended for use with the -r option.

Location

This command is provided by the `IKEv2` module when `IKEv2` is defined.

ipsec policy

Command Name

`ipsec policy - add, delete, and view IKE and IPSEC policy database information`

Syntax

- `ipsec policy {-n -a [-c] -d -e -g [-h] [-k <integer>] [-l] -m -n -p -r -s -t [-y]}`
- `ipsec policy {-v [-n <policy name>]}`
- `ipsec policy {-x <policy name>}`

Parameters

`-a`	authentication protocol list
`-c`	enable/disable inclusion of TFC padding in outgoing packets -- optional
`-d`	destination id
`-e`	encryption protocol list
`-g`	remote name
`-h`	short ICV length (for SHA2-xxx family) -- optional
`-k`	IPSEC SA lifetime -- default is 3600, applies only when used with IKE
`-l`	priority (1(highest)...7(lowest)) -- default is 4
`-m`	IPsec mode (transport or tunnel)
`-n`	policy name
`-p`	upper-layer protocol -- one of udp, tcp, icmp, icmp6, or any
`-r`	remote endpoint
`-s`	source id string
`-t`	SA type -- AH, ESP, or AHESP
`-v`	view policy -- all policies will be displayed unless used with -n
`-x`	delete policy with given name or ALLPOLICIES
`-y`	enable transmission of dummy packets -- optional

Description

This command will add, view, and display IKE and IPSEC policy database information.

Notes/Status

- The -x option will delete a given policy or all policies if "-x ALLPOLICIES" is used.
- The -v option will display a given policy or all policies if no -n parameter is given.
- Higher priority policies (as indicated by the -l option) will be used when the packet matches more than one policy. For example, a policy to match and bypass UDP traffic to port 500 (the IKE port) should typically be given higher priority than a policy to secure UDP traffic to other ports. This allows the IKE protocol which uses port 500 to establish SAs for other UDP traffic. The bypass policy for IKE traffic is installed by default at priority level 2 when IKE is enable.

Example CLI commands (note: the commands must be on a single line)

- ESP-only, transport mode, authentication (SHA1), encryption (3DES), udp protocol, IPv4

```
ipsec policy -n Policy1 -g mongo -c -h -l 4 -p udp -m transport -t ESP
      -a HMAC-SHA-1 -e 3DES-CBC -r 10.0.0.76 -s 10.0.0.140 -d 10.0.0.76
```

- ESP-only, transport mode, authentication (MD5 or SHA1) and encryption (NULL or 3DES or AES128), udp protocol, IPv4

```
ipsec policy -n Policy1 -g mongo -c -h -l 4 -p udp -m transport -t ESP
      -a HMAC-MD5,HMAC-SHA-1 -e NULL_ENC,3DES-CBC,AES128-CBC -r 10.0.0.76

      -s 10.0.0.140 -d 10.0.0.76
```

- ESP-only, tunnel mode, encryption (3DES) and authentication (MD5), tcp protocol, IPv4

```
policy -n Policy2 -g mongo -t ESP -a HMAC-MD5 -e 3DES-CBC -m transport

       -p tcp -r 10.0.0.76 -s 10.0.0.140 -d 10.0.0.76
```

- ESP-only, transport mode, encryption (3DES) and authentication (MD5), udp protocol, IPv6

```
policy -n Policy1V6 -g mongoV6 -t ESP -a HMAC-MD5 -e 3DES-CBC -m transport

       -p udp -r fe80::0240:f4ff:feed:8b77
       -s fe80::862b:2bff:fe88:2662 -d fe80::0240:f4ff:feed:8b77
```

- ESP-only, transport mode, encryption (AES128) and authentication (SHA), tcp protocol, IPv6

```
policy -n Policy2V6 -g mongoV6 -t ESP -a HMAC-SHA-1 -e AES128-CBC
       -m transport -p tcp -r fe80::0240:f4ff:feed:8b77
       -s fe80::862b:2bff:fe88:2662 -d fe80::0240:f4ff:feed:8b77
```

Location

This command is provided by the `IPsec` module when `IPSEC` and `IKEv2` are defined.

ike reload

Command Name

`ike reload - reloads configuration and restarts ike`

Syntax

`ike reload`

Parameters

None

Description

This command flushes IKEv1 Phase 1 and Phase 2 SAs and IKEv2 SAs, and remote configuration database. Then it restarts IKE reading the configuration command file.

Notes/Status

The IKE configuration file is defined in ipport.h, IKE_SCRIPT_FILE as ike_rc by default. This file can contain any iniche commands and is executed after IKE starts and after a small delay to allow interfaces to acquire IP addresses.

Location

This command is provided by the `IKEv2` module when `IKEv2` is defined.

ike remote

Command Name

`ike remote - add, delete, and view IKE remote peer configuration information`

Syntax

- `ike remote {-n -a [-b] [-c] -d -e -f -g [-h] [-i] [-l] -m -n -p -r -s [-u] [-w] -z}`
- `ike remote {-v [-n <remote name>]}`
- `ike remote {-x <remote name>}`

Parameters

`-a`	authentication protocol list, ike version 1 supports • MD5 • SHA and ike version 2 supports • HMAC-MD5 • HMAC-SHA-1 • HMAC-SHA-2-256 • HMAC-SHA-2-384 • HMAC-SHA-2-512
`-b`	peer's X.509 certificate
`-c`	local end's X.509 certificate(s) (to be sent to remote peer)
`-d`	dh group algorithm list • MODP768 • MODP1024 • MODP1536 • MODP2048 • MODP3072 • MODP4096 • MODP6144 • MODP8192

`-e`	encryption protocol list • 3DES-CBC • AES128-CBC • AES192-CBC • AES256-CBC • NULL_ENC
`-f`	identifier (expected from remote peer)
`-g`	identifier (to be sent to remote peer)
`-h`	the filename that contains the passphrase for PSK
`-i`	interval_to_send -- retransmission interval
`-l`	IKE SA lifetime
`-m`	auth method algorithm • PresharedKey • RSASIG
`-n`	remote name
`-p`	prf algorithm list • HMAC-MD5 • HMAC-SHA-1 • HMAC-SHA-2-256 • HMAC-SHA-2-384 • HMAC-SHA-2-512
`-r`	remote endpoint
`-u`	accept this version (can be configured to accept both ikev1 and ikev2)
`-v`	view remote -- all policies will be displayed unless used with -n
`-w`	initiate this version (only one version can be initiated, ikev1 or ikev2)
`-x`	delete remote with given name or ALLREMOTES
`-z`	version to configure in this command

Description

This command will add, delete, and view IKE remote peer configuration information. The parameters are used to negotiate the algorithms to be used to establish secure communications for the exchange of IPSEC parameters and algorithms. The algorithms used by IKE could be different than those negotiated for use by IPSEC. In particular the IKEv1 authentication algorithms are more limited than the IKEv2 algorithms.

Notes/Status

- The -x option will delete a given remote or all remotes if "-x ALLREMOTES" is used.
- The -v option will display a given remote or all remotes if no -n parameter is given.
- Deleting a remote peer does not delete any Security Association (SAs) currently in use.

Example CLI commands.

These commands could appear together as they represent logically different remotes. The first two examples configure both IKEv1 and IKEv2 to the same remote peer, but only IKEv2 would be initiated to the remote while either IKEv1 or IKEv2 would be accepted. Notice that the algorithm lists can be different. Example 3 and 4 are similar but use IPv6 addresses. The names of the remotes are only used internally and do not reflect DNS name. Each policy command must reference a remote configured with the remote command.

- IKEv2 [-z 2]: initiate using IKEv2 [-w] and accept, that is respond to, IKEv2 [-u], use presharedkey [-m PSK] with the shared secret "secret" [-h secret]

```
remote -n mongo -z 2 -w -u -a HMAC-MD5 -e 3DES-CBC -d MODP2048,MODP1024

       -p HMAC-MD5 -m PSK -h secret
       -r 10.0.0.76 -f ipv4:10.0.0.76 -g ipv4:10.0.0.140
```

- IKEv1 [-z 1]: do not initiate using IKEv1 [there is no -w] and accept, that is respond to, IKEv1 [-u], use presharedkey [-m PSK] with the shared secret "secret" [-h secret]

```
remote -n mongo -z 1    -u -a SHA -e 3DES -d MODP1024 -p MD5 -m PSK
       -h secret -r 10.0.0.76 -f ipv4:10.0.0.76 -g ipv4:10.0.0.140
```

- IKEv2 [-z 2]: initiate using IKEv2 [-w] and accept, that is respond to, IKEv2 [-u], use presharedkey [-m PSK] with the shared secret "secret" [-h secret]

```
remote -n mongoV6 -z 2 -w -u -a HMAC-MD5 -e 3DES -d MODP2048,MODP1024
       -p MD5 -m PSK -h secret
       -r fe80::0240:f4ff:feed:8b77 -f ipv6:fe80::0240:f4ff:feed:8b77
       -g ipv6:fe80::862b:2bff:fe88:2662
```

- IKEv1 [-z 1]: do not initiate using IKEv1 [there is no -w] and accept, that is respond to, IKEv1 [-u], use presharedkey [-m PSK] with the shared secret "secret" [-h secret]

```
remote -n mongoV6 -z 1    -u -a SHA -e 3DES -d MODP1024 -p MD5 -m PSK
       -h secret -r fe80::0240:f4ff:feed:8b77
       -f ipv6:fe80::0240:f4ff:feed:8b77
       -g ipv6:fe80::862b:2bff:fe88:2662
```

Location

This command is provided by the `IKEv2` module when `IKEv2` is defined.

ike commands

Command Name

`ike commands - print the configuration in the form of command lines`

Syntax

```
ike commands [-f filename]
```

Parameters

`-f`	filename for output

Description

This command prints the running configuration of "ike remote" and "ipsec policy" configuration commands. The commands will be formatted so that they could be used as input at startup, for example using the IKE_SCRIPT_FILE defined in ipport.h, ike_rc by default.

Notes/Status

- #define IKE_DUMP_CMD_FILE "ike_dump_cmds.txt"

Location

This command is provided by the `IKEv2` module when `IKEv2` is defined.

ike debug

Command Name

`ike debug - enable IKE debug logging and tracing`

Syntax

```
ike debug [-t] [-d]
```

Parameters

`-t`	turn on ike debug tracing
`-d`	turn on ike debugging

Description

This command enables or disables IKE related debugging output. If no parameters are provided debugging is turned off.

Notes/Status

- The image must be built with IKE_DEBUG_TRACE enabled in ipport.h for "-t" to be available.
- The image must be built with IKE_DEBUG enabled in ipport.h for "-d" to be available.

Location

This command is provided by the `IKEv2` module when `IKEv2` is defined.

4.9 Compile-time considerations

The following `#defines` are located in `ipport.h` and may be enabled/disabled to suit your requirements.

IKE_DEBUG		turns on debugging output
IKE_DEBUG_TRACE		turns on more debugging output, especially packet tracing
IKEv2_MENUS		Enables the IKEv2 CLI commands
IKE_ADD_TEST_BYPASS_POLICIES		adds bypass policies that let all traffic through which does not match other policies. It should only be used when testing in non-production environments
IKE_SCRIPT_FILE	"ike_rc"	Specifies the file to be read during ike initialization. May contain commands like "ipsec policy" and “ike remote”. These commands can be very long so it is helpful to put them in this file.
IKE_DUMP_CMD_FILE	"ike_dump_cmds.txt"	Specifies the file to be written when the CLI "ike commands" is executed which writes the running config to a file.
MAXIOSIZE	512	Override default because IKE has potentially long command lines which could be written using the "ike commands" CLI
MAXCMD	512	Override default because IKE commands can be very long

Also note that ipsec/ipsecapi.h contains:

```
/* total time (in seconds) to wait for IPv6 to complete its initialization */
#define IPV6_ADDR_CFG_CHK_WAIT_TIME     60
```

5 Examples: Securing communications between two InterNiche-based systems

The first example secures a TCP connection using IPSec and IKE in transport mode, using a Pre-Shared Key. The second example involves ICMP with IPSEC and IKE in tunnel mode, using X.509 certificate based authentication.

Before we begin:

1. Verify that extras/ike_rc resides in the port directory (ReferencePorts/w32_nichetask_vs)
2. Issue the "ipsec netstat" command from a CLI and verify that a UDP bypass policy on port 500 is configured

Additional notes concerning these examples:

- "host1" has IPv4 address 10.0.0.77
- "host2" has IPv4 address 10.0.0.78
- "`secret_file.txt`" has been copied from the "extras" distribution directory into the reference port's executable directory on each system

5.1 Example 1: TCP connection in transport mode using a Pre-Shared Key with IPSec and IKEv2

Step 1: Configure hosts and verify pre-connection status

On "host1":

Issue the security policy and the ike remote command for tcp traffic:

```
-> ipsec policy -n policy1 -g remoteINFO -t ESP -e 3DES-CBC -a HMAC-SHA-2-384 -m transport -p
tcp -r 10.0.0.78 -s 10.0.0.77 -d 10.0.0.78
-> ike remote -n remoteINFO -z 2 -w -u -a HMAC-SHA-2-384 -e 3DES-CBC -p HMAC-SHA-2-384 -m
PresharedKey -d MODP768 -r 10.0.0.78 -f ipv4:10.0.0.78 -g ipv4:10.0.0.77 -h secret_file.txt
```

On "host2":

```
-> ipsec policy -n policy1 -g remoteINFO -t ESP -e 3DES-CBC -a HMAC-SHA-2-384 -m transport -p
tcp -r 10. 0.0.77 -s 10.0.0.78 -d 10.0.0.77
-> ike remote -n remoteINFO -z 2 -w -u -a HMAC-SHA-2-384 -e 3DES-CBC -p HMAC-SHA-2-384 -m
PresharedKey -d MODP768 -r 10.0.0.77 -f ipv4:10.0.0.77 -g ipv4:10.0.0.78 -h secret_file.txt
```

Step 2: Verify policy configuration

On both "host1" and "host2" issue the “ipsec netstat” CLI command and verify that the desired remote, source and destination IP addresses are displayed:

```
-> ipsec netstat
spid   pri proto policy  src-id              dst-id              mode     gateway
1      2   udp   bypass  10.0.0.78:500       any
7      4   tcp   secure  10.0.0.78           10.0.0.77           transport 10.0.0.77
23     7   0     bypass  any                 any
```

Step 3: Establish a TCP connection and verify link status

Using any TCP application (Telnet, FTP, TCP Echo, etc) establish a link between the systems and then use the "ipsec netstat" CLI command to verify the secure link. The output should resemble this:

```
-> ipsec netstat
IPSecSession SPI=b96e76e3, SPID=7
     CreateTime=603, Life=3600 secs, HardExpiry in 3439 secs, SoftExpiry in 3429 secs
ESP  Outbound SA, SPI=d17c6fc7
     HMAC-Auth: SHA384
             Key: [len=48]
2805572ebb343a19455d24854f69ff9fdc722fe56d737c85505ffdde66a431cf65a21442d160383d19f202e7e4b7bc6a
     Encryption: 3DES
             Key: [len=24] c176e39efd2a1fdadcae2c3bfb5416e55b259e34c70186c7
Dummy pkts transmitted: 0
ESP  Inbound  SA, SPI=b96e76e3
     HMAC-Auth: SHA384
             Key: [len=48]
7ccc4ee9cd241e671e28de3d74bfbd9a12286f2142ff0215855e78845e34d04686027c400b7be700600f2494197155f9
     Encryption: 3DES
             Key: [len=24] 7aba94f4cbb051feb9750457ba07d5eaa720d34fa7a27c1c
Dummy pkts received: 0
spid   pri proto policy  src-id              dst-id              mode     gateway
1      2   udp   bypass  10.0.0.78:500       any
7      4   tcp   secure  10.0.0.78           10.0.0.77           transport 10.0.0.77
23     7   0     bypass  any                 any
```

Notes:

- Use of a network traffic capture program such as WireShark should show IKE_SA_INIT and IKE_AUTH packets in both directions, followed by an ESP protocol traffic
- Possible reasons for an unsuccessful traffic:
    - No IKE_AUTH in the Wireshark trace – verify remote host IP address is correct
    - Verify traffic selector type (tcp, udp, icmp) match on both "ipsec policy" commands
    - Verify the algorithms used match.

5.2 Example 2: ICMP connection in tunnel mode using X.509 certificate-based authentication

Step 1: Configure hosts and verify the pre-connection status

On "host1":

```
-> ce x509 -a cacert.pem -d 1
-> ce x509 -a supamcert.pem -d 1 -k supamkeyc.pem
-> ike remote -n dexter -z 2 -w -u -a HMAC-SHA-2-512 -e AES192-CBC -p HMAC-SHA-2-256 -m RSASIG -
d MODP3072 -r 10.0.0.78 -f asn1dn: -g asn1dn: -b supascert.pem -c supamcert.pem,cacert.pem
-> ipsec policy -n policy0 -g dexter -t ESP -e 3DES-CBC -a HMAC-SHA-2-512 -m tunnel -p icmp -r
10.0.0.78 -s 10.0.0.77 -d 10.0.0.78 -h
```

On "host2":

```
-> ce x509 -a cacert.pem -d 1
-> ce x509 -a supascert.pem -d 1 -k supaskeyc.pem
-> ike remote -n dexter -z 2 -w -u -a HMAC-SHA-2-512 -e AES192-CBC -p HMAC-SHA-2-256 -m RSASIG -
d MODP3072 -r 10.0.0.77 -f asn1dn: -g asn1dn: -b supamcert.pem -c supascert.pem,cacert.pem
-> ipsec policy -n policy0 -g dexter -t ESP -e 3DES-CBC -a HMAC-SHA-2-512 -m tunnel -p icmp -r
10.0.0.77 -s 10.0.0.78 -d 10.0.0.77 -h
```

Step 2: Verify the policy configuration

On both "host1" and "host2" issue the “ipsec netstat” CLI command and verify that the desired remote, source and destination IP addresses are displayed.

Step 3: Execute

From either side, "ping" the other and verify operational status using "ipsec netstat"

A final note about these examples: To avoid errors caused by mis-entering these long commands, it may be wise to enter them into one or more text files which are then invoked through the CLI using the "call" command. Remember to clear the entries (or restart the target) between attempts. Alterhatively, these commands can be place in the `ike_rc` file which will be read at startup (depending on your system configuration).

6 Porting to non-InterNiche TCP/IP stacks

IPSec toolkit is designed to be portable and high performance. In order to provide for easy integration of the IPSec toolkit with different network kernels, the buffers used by the network stack must be converted to buffers used by the IPSec toolkit and then back to the network stack buffer. If this conversion is not efficient, then performance will suffer.

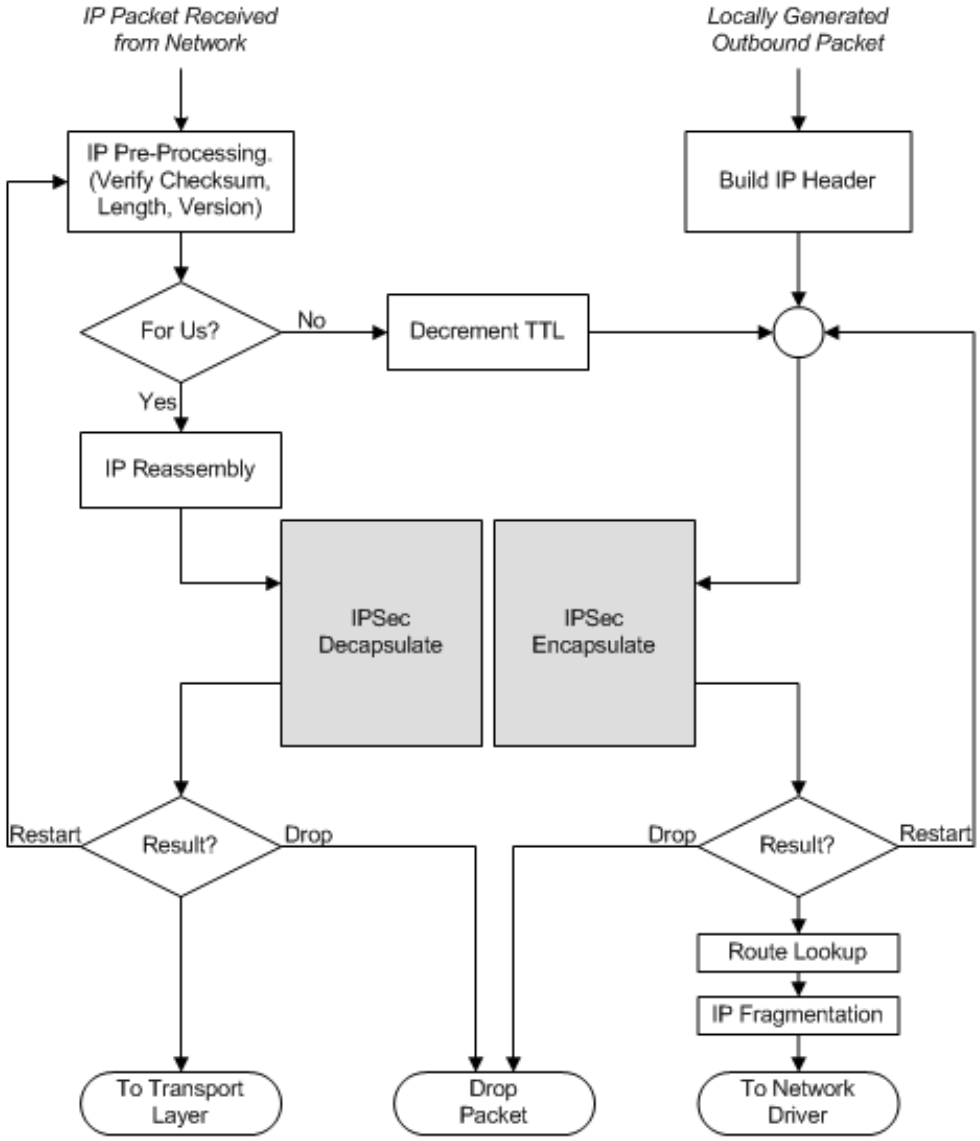
In order to not degrade performance, the IPSec toolkit uses a packet abstraction that allows different packet and buffer structures to be used without any changes to the source code. All accesses to the packet buffers are through macros. Different buffer structures can be easily defined using macros to map the packet abstraction with the network buffers that is used by the network stack.

The packet buffer abstraction allows for packets to be fragmented into a number of buffers. Many networking kernels, such as NetBSD, FreeBSD, VxWorks, etc. use fragmented packets. There is no assumption made as to the minimum size of the fragments or the number of fragments. However, the buffers must be large enough so that all headers can fit in a single buffer.

IPSec is designed to port easily to any operating system and any TCP/IP protocol stack. Integrating the IPSec Toolkit with the TCP/IP stack requires only a few steps. One of the requirements for The IPSec Toolkit is that you need to have source code access to the TCP/IP stack. However, most TCP/IP stacks will provide hooks to call third-party networking software. Examples of such hooks are the netfilter hooks in Linux and similar hooks in VxWorks. In such cases, The IPSec Toolkit can be integrated without requiring source code for the TCP/IP stack.

6.1 IP Packet Flow

The following flowchart shows a typical IP protocol stack processing. Integrating The IPSec Toolkit into such a protocol stack involves adding hooks in a couple of locations within the stack as shown in the flowchart.



As the flowchart shows, IPSec encapsulation is performed for outbound traffic by calling the PacketEncapsulate function. Similarly, IPSec decapsulation is performed for inbound traffic that is addressed to the system by calling the PacketDecapsulate function. The output packet from these functions is then processed as specified by the result code.

6.2 Porting Overview

Here are a few quick steps to porting and integration of The IPSec Toolkit with the TCP/IP protocol stack:

System Dependencies

Tuning and platform specific definitions can be found in the two include files "ipsecconf.h" and "sysdep.h", respectively and may be modified with a text editor. In addition to making minor changes to these files, you should also check the Makefile and select proper build options.

IPSec Configuration

IPSec configuration, such as the maximum number of policies and security associations supported on your platform, is specified by setting parameters in the file ipsecconf.h. This is important to set these correctly since it determines the amount of data space used by IPSec. You will have to specify the number of security associations (SA) and the number of security policies (SP).

Packet Buffers

Most packet buffers can be mapped to the portable buffer mechanism included with IPSec. In most cases, the mapping is done through macros. Hence IPSec will be able to use stack dependent packet buffers without any buffer conversion.

The `pkt_iniche.h` and `pkt_iniche.c` files are used to "map" an IPsec packet buffer to the InterNiche TCP/IP stack buffer. These two files can be used as a template for your implementation-specific buffering scheme.

Initialization

The IPSecInit() function MUST be called once before any other function is called. This function performs any required initialization.

The Packet Interface

IPSec interfaces to the TCP/IP stack. IPSec requires that all packets (both inbound as well as outbound) are passed to IPSec.

The file "ipsecapi.h" contains the definitions and function prototypes of the Packet Interface functions

For inbound packets, you would normally perform IP pre-processing before passing the packet to the IPSec module. Pre-processing operations include validating length and checksum fields in the IP header, re-assembling fragments (only required for packets addressed to the system), and handling any IP options that need to be processed. Two cases need to be considered here: (a) packets addressed to the system and (b) packets that need to be forwarded. In the case of (a), the packet is then passed to PacketDecapsulate() function. Note that all packets MUST be passed to this function, not just packets where the protocol is AH or ESP. This is to ensure that policies are being enforced. In the case of (b), the PacketEncapsulate() function must be called AFTER the TTL is decremented by 1. Again, all forwarded packets must be passed to PacketEncapsulate() function.

For outbound packets, PacketEncapsulate must be called AFTER the IP header is filled in. The IP checksum field will be recalculated by IPSec and hence can be set to be zero. However, if it's not set to zero, then it must be a proper checksum value. Implementations must ensure that forwarded packets don't get passed to IPSec twice.

After all IPSec processing completed on the packet, the callback functions EncapsulateDone() and DecapsulateDone() will be called. At this time, the TCP/IP stack can continue processing on the packet. Sample implementations of these functions are provided in the file "callbacks.c". Implementers may modify this file accordingly.

OS interaction

IPSec/IKE assumes (a) module; (b) gio; (c) CryptoEngine.

TCP-IP

If an IP address has changed due to IPv6 routers appearing/leaving, it will be necessary to issue the "ike reload" command or api. This will clear all ipsec policies and ike information, then recreate ipsec bypass policies, then reread the ike_rc file to execute the additional "ipsec policy" commands and "ike remote" commands.