



Treck IPsec User Manual



Complete Internet Solutions

Technical Support:

5041 Lamart Drive #240 Riverside, California 92507
Phone: (909) 787-7056 Fax: (909) 787-8803

Corporate Headquarters:

Treck, Inc. 431 Ohio Pike Suite 210 North Cincinnati, Ohio 45255
Phone: (513) 528-5732 Fax: (513) 528-5740

Contents

1 Introduction to IPsec/IKE	5
1.1 Authentication Header (AH)	5
1.2 Encapsulating Security Payload (ESP)	7
1.3 Internet Key Exchange (IKE)	10
1.3.1 Phase One Negotiation Attributes	10
1.3.1.1 PROTOCOLIDENTIFIERANDTRANSFORMIDENTIFIER	10
1.3.1.2 ENCRYPTIONALGORITHM	10
1.3.1.3 HASHALGORITHM	10
1.3.1.4 AUTHMETHOD	10
1.3.1.5 DHGROUPDESCRIPTION	10
1.3.1.6 LIFETYPE	10
1.3.1.7 LIFE DURATION	11
1.3.1.8 PSEUDORANDOMFUNCTION	11
1.3.1.9 KEYLENGTH	11
1.3.2 Phase Two Negotiation Attributes	11
1.3.2.1 PROTOCOLIDENTIFIERANDTRANSFORMIDENTIFIER	11
1.3.2.2 LIFETYPE	11
1.3.2.3 LIFE DURATION	12
1.3.2.4 DHGROUPDESCRIPTION	12
1.3.2.5 ENCAPSULATIONMODE	12
1.3.2.6 AUTHENTICATIONALGORITHM	12
1.3.2.7 KEYLENGTH	12
1.3.2.8 KEY ROUNDS	12
1.3.3 Treck IKE features	13
2 Security Policy Database (SPD) and Security Association Database (SAD)	14
2.1 SPD	14
2.2 SAD	15
2.3 Outbound Processing	15
2.4 Inbound Processing	16
2.5 Treck SPD and SAD features	16
3 Settings in trsystem.h	17
3.1 TM_USE_IPSEC	17
3.2 TM_USE_IKE	17
3.3 TM_IPSEC_DEBUG	17
3.4 TM_IKE_DEBUG	17
3.5 TM_IPSEC_USE_ANTIPLAY	17
3.6 TM_IPSEC_INCOMING_ICMP_BYPASS	17
3.7 TM_IPSEC_INCOMING_ICMP_NOSRCCHK	17
3.8 TM_6_IPSEC_ICMPV6_NDMLD_BYPASS	17
3.9 TM_IPSEC_DEFAULT_REPLAY_SIZE	17
3.10 TM_IKE_DHSECRET_DEFAULT_SIZE	17
3.11 TM_IKE_PFS_KEY_ENABLED	17
3.12 TM_IKE_PHASE1_AGGRESSIVE_ENABLED	18
3.13 TM_IKE_AGGRESSIVE_DHGROUP	18
3.14 TM_IKE_INITIAL_CONTACT_ENABLED	18
3.15 TM_IKE_USE_DEFAULT_PRESHARED_KEY	18
3.16 TM_IPSEC_BYPASS_NESTIKE_PACKET	18
3.17 TM_IPSEC_USE_SW_CRYPTENGINE	18
3.18 TM_IPSEC_USE_HF7951_CRYPTENGINE	18
3.19 TM_USE_PKI	18
3.20 TM_PKI_CHECK_CERT_ALIVE	18
3.21 TM_PKI_CERT_NOT_VERIFY	18
3.22 TM_USE_RIPEMD	18

3.23	TM_USE_DES	18
3.24	TM_USE_3DES	18
3.25	TM_USE_BLOWFISH	18
3.26	TM_USE_RC5	18
3.27	TM_USE_CAST128	19
3.28	TM_USE_AES	19
3.29	TM_USE_TWOFISH	19
3.30	TM_PUBKEY_USE_DIFFIEHELLMAN	19
3.31	TM_PUBKEY_USE_RSA	19
3.32	TM_PUBKEY_USE_DSA	19
4	Function Reference	20
4.1	IPsec and IKE initialization	20
4.1.1	tfUseIpsec	20
4.1.2	tfStartIke	22
4.1.3	tfStartEnhancedIke	24
4.1.4	tfIpsecUninitialize	26
4.1.5	tfIpsecSetOptions	27
4.2	Pre-shared Key Management API	29
4.2.1	tfPresharedKeyAdd	29
4.2.2	tfPresharedKeyDelete	30
4.2.3	tfPresharedKeyClear	31
4.3	PKI API	32
4.3.1	tfUsePki	32
4.3.2	tfPkiUninitialize	32
4.3.3	tfPkiCertificateAdd	33
4.3.4	tfPkiCertificateDelete	35
4.3.5	tfPkiCertificateClear	36
4.3.6	tfPkiOwnKeyPairAdd	37
4.3.7	tfPkiPubKeyAdd	38
4.4	ISAKMP Policy Management API	39
4.4.1	tfIkeAddPhase1Transform	39
4.4.2	tfIkeDeletePhase1Transform	40
4.5	IPsec Policy Database Management API	41
4.5.1	tfPolicyRestore	41
4.5.2	tfPolicyAdd	46
4.5.3	tfPolicyAddBundle	48
4.5.4	tfPolicyDelete	49
4.5.5	tfIpsecPolicyQueryBySelector	50
4.5.6	tfIpsecPolicyQueryByIndex	52
4.5.7	tfPolicyClear	53
4.6	SA Database Management API	54
4.6.1	tfSadbRecordGenerate	54
4.6.2	tfSadbRecordDelete	56
4.6.3	tfSadbRecordDeleteByPolicy	57
4.6.4	tfSadbRecordDeleteByDestination	58
4.6.5	tfSadbRecordClear	59
4.6.6	tfSadbRecordGet	60
4.6.7	tfSadbRecordFind	61
4.6.8	tfSadbRecordSetOptions	63
4.6.9	tfSadbRecordManualAdd	64
4.7	Log Function API	66
4.7.1	tfUseIpsecLogging	66
4.7.2	tfIpsecLogWalk	67
4.8	Use Hardware Accelerator	68
4.8.1	tfCryptoEngineRegister	68
4.8.2	tfCryptoEngineDeRegister	70
4.8.3	tfCryptoEngineAddAlgorithm	71

5 Structure Reference	73
5.1 Cryptology Structures	73
5.1.1 ttGenericKey	73
5.1.2 ttAhAlgorithm	74
5.1.3 ttEspAlgorithm	76
5.2 Policy Management Structures	77
5.2.1 ttIpssecSelectorInString	77
5.2.2 ttIpssecSelector	80
5.2.3 ttPolicyContentInString	81
5.2.4 ttPolicyContent	84
5.2.5 ttIpssecPolicyPair	85
5.2.6 ttPolicyEntry	87
5.3 SA Management Structures	88
5.3.1 ttSadbRecord	88
5.3.2 ttSaIdentity	91
5.3.3 ttChildSaInfo	92
5.4 Crypto Engine Structures	93
5.4.1 ttCryptoSessionOpenFuncPtr	93
5.4.2 ttCryptoSessionFuncPtr	93
5.4.3 ttCryptoEngineInitFuncPtr	93
5.4.4 ttCryptoGetRandomWordFuncPtr	94
5.4.5 tsCryptoEngine	94
6 Examples	96
6.1 Manual keying	96
6.1.1 Define the Policy	96
6.1.2 Set trsystem.h	96
6.1.3 Start Treck	97
6.1.4 Add driver interface below Ethernet link layer	97
6.1.5 Call tfUseIpssec	97
6.1.6 Add policy and SA	98
6.1.7 Open the added interface	98
6.1.8 Test case	98
6.1.9 Procedure addPolicyAndSa	98
6.1.10 Procedure testIpssecDifferentPolicy	102
6.2 IKE (Automatic keying)	103
6.2.1 Define the Policy	103
6.2.2 Set trsystem.h	103
6.2.3 Start Treck	104
6.2.4 Add driver interface below Ethernet link layer	104
6.2.5 Call tfUseIpssec	105
6.2.6 Add policy	105
6.2.7 Open the added interface	105
6.2.8 Call tfStartIke	105
6.2.9 Add Preshared-key	106
6.2.10 Test case	106
6.2.11 Procedure addPolicy	106
6.2.12 Procedure testIpssecDifferentPolicy	108
7 References	110

1 Introduction to IPsec/IKE

IPsec is a set of extensions to the IP protocol family. It provides services allowing for authentication, integrity, and confidentiality. Unlike SSL, which provides security services over TCP/IP, IPsec provides security services at the network layer so that it is transparent to the IP applications. IPsec is optional to the IPv4 stack but is a mandatory part of the IPv6 stack.

IPsec uses two protocols to provide traffic security – Authentication Header (AH) and Encapsulating Security Payload (ESP). A Security Association (SA) affords these security services to the traffic carried by it. AH and ESP may run in either of the two modes, transport mode or tunnel mode.

Internet Key Exchange (IKE) negotiates properties of SA between peers.

The Treck IPsec and IKE features can be found at each of the following sections.

1.1 Authentication Header (AH)

AH comes after the basic IP header and contains cryptographic hashes of the data and identification information. It is used to provide data integrity, data origin authentication, and optional anti-replay services to IP.

AH has been assigned the IP protocol number 51. The header format of AH is shown in Figure 1.

Next Header	Payload Length	Reserved
Security Parameter Index (SPI)		
Sequence Number		
Authentication Data (For IPsec, this field is 96 bits)		

Figure 1 AH Header Format

Figure 2 illustrates AH positioning for a typical IP packet, on a before and after basis, for both transport mode and tunnel mode. (a) is for IPv4 packet, and (b) is for IPv6 packet

Original IPv4 Header (any options)	TCP	Data
---------------------------------------	-----	------

IPv4----After applying AH (transport mode)

Original IPv4 Header (any options)	AH	TCP	Data
---------------------------------------	----	-----	------

IPv4----After applying AH (tunnel mode)

New IPv4 Header (any options)	AH	Orig IPv4 Header (any options)	TCP	Data
----------------------------------	----	-----------------------------------	-----	------

(a)

IPv6----Before applying AH

Original IPv6 Header	Extension Header if present	TCP	Data
----------------------	-----------------------------	-----	------

IPv6----After applying AH (transport mode)

Original IPv6 Header	Extension Header if present *	AH	Dest. *	TCP	Data
----------------------	-------------------------------	----	---------	-----	------

*: If destination extension header presents, it could be before AH, after AH or both

IPv6----After applying AH (tunnel mode)

New IPv6 Header	New Extension Headers if present	AH	Orig IPv6 Header	Extension Headers if present	TCP	Data
-----------------	----------------------------------	----	------------------	------------------------------	-----	------

(b)

Figure 2 AH Position In IPv4 and IPv6

For either transport mode or tunnel mode, the AH header authenticates the whole IP packet except for the mutable fields in outside IP header. For the definition of mutable and immutable fields, please refer to RFC 2401 and RFC 2402.

Treck AH provides the following features:

- Supports HMAC-MD5-96 and HMAC-SHA1-96, HMAC-RIPMD160-96 and NULL authentication algorithm
- Tunnel and transport mode
- Nested tunnel
- SA bundle (with ESP)
- Easy to add or remove algorithms
- Supports both IPv4 and IPv6.

1.2 Encapsulating Security Payload (ESP)

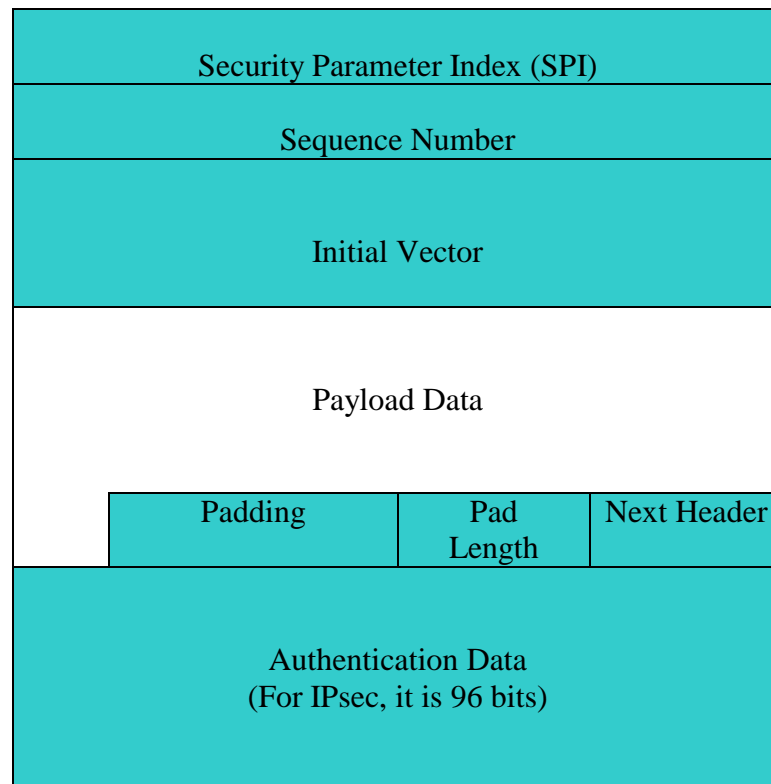


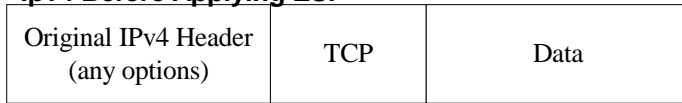
Figure 3 ESP

ESP allows rewriting of the payload in encrypted form. It provides confidentiality, data origin authentication, anti-replay, and data integrity services to IP. ESP does not apply to IP header fields preceding it.

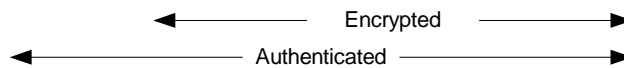
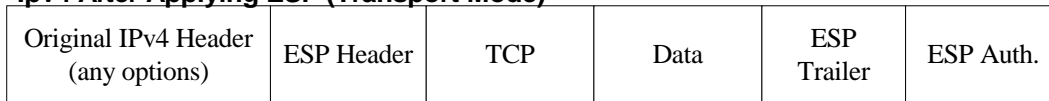
ESP has been assigned the IP protocol number 50. If a NULL encryption algorithm is used, there is no 'Initial Vector' field. Additionally, if a NULL authentication algorithm is used, there is no 'Authentication Data' field. However, encryption algorithm and authentication algorithm may not both be NULL. The padding length may be from 0 to 255 bytes. However, Treck IPsec, like most others, uses minimum padding.

Figure 4 illustrates ESP positioning for a typical IP packet on a 'before-and-after basis' for both transport and tunnel mode. Diagram (a) is for IPv4 packet and (b) is for IPv6 packet.

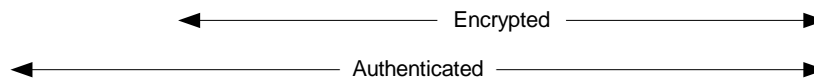
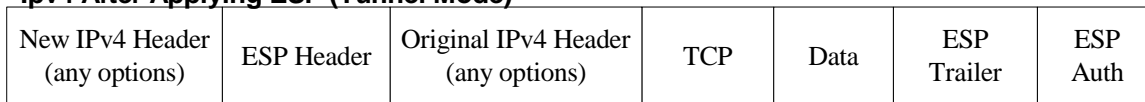
Ipv4 Before Applying ESP



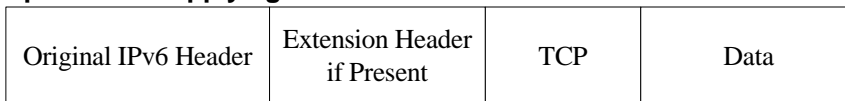
Ipv4 After Applying ESP (Transport Mode)



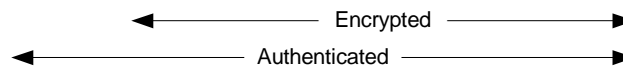
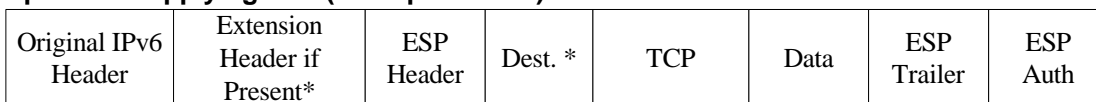
Ipv4 After Applying ESP (Tunnel Mode)



Ipv6 Before Applying ESP



Ipv6 After Applying ESP (Transport Mode)



* If destination header presents, it could be before ESP, after ESP, or both

Ipv6 After Applying ESP (Tunnel Mode)

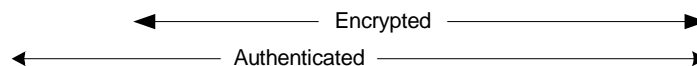
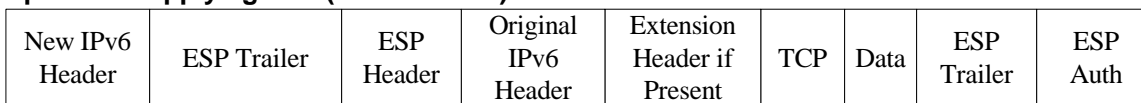


Figure 4 ESP Position In IPv4 and IPv6

Treck's ESP provides the following features:

- Supports NULL, DES, 3DES (Both DES and 3DES has configurable weak key checking and parity bit checking), BLOWFISH (key size 40-448 bits), CAST128 (key size 40-128 bits), RC5 (key size 40-2040 bits), IDEA (key size 128 bits), AES (RIJNDAEL, key size 128, 192, 256 bits), TWOFISH (key size 128,192, 256 bits) algorithm and in CBC mode. RC5 and IDEA algorithms require patent license.
- Supports HMAC-MD5-96, HMAC-SHA1-96, HMAC-RIPEMD160-96 and NULL authentication algorithm
- Tunnel and transport mode
- Nested tunnel
- SA bundle (with AH)
- Minimum padding
- Easy to add or remove algorithms
- Supports IPv4 and IPv6

1.3 Internet Key Exchange (IKE)

IKE negotiates the SA properties between peers. IKE includes two phases. There are two modes for the phase one negotiation, *main mode* and *aggressive mode*, and only one mode for phase two, *quick mode*.

The phase one negotiation results in a secure channel, i.e. ISAKMP SA, between peers. All the following negotiations will be protected in this channel. IKE phase two negotiates IPsec SA properties.

1.3.1 Phase One Negotiation Attributes

IKE is a negotiation protocol. Treck IKE supports the following attributes, per RFC 2409.

1.3.1.1 PROTOCOL IDENTIFIER AND TRANSFORM IDENTIFIER

The phase 1 Protocol Identifier is

TM_DOI_PROTO_ISAKMP 1

The only transformer ID for this protocol is

TM_DOI_KEY_IKE 1

1.3.1.2 ENCRYPTION ALGORITHM

TM_IKE_DES_CBC 1

TM_IKE_IDEA_CBC 2

TM_IKE_BLOWFISH_CBC 3

TM_IKE_RC5_R16_B64_CBC 4

TM_IKE_3DES_CBC 5

TM_IKE_CAST_CBC 6

TM_IKE_AES_CBC 7

The above algorithms are specified in RFC 2409. AES is specified in RFC 3602. The RC5 and IDEA algorithms are subject to patent protection, and therefore Treck does not provide source code for these algorithms. For Treck IKE, you may also negotiate TWOFISH. Interoperation with other implementations is not guaranteed.

TM_IKE_TWOFISH_CBC 13

1.3.1.3 HASH ALGORITHM

TM_IKE_MD5 1 /*MD5 hash algorithm*/

TM_IKE_SHA1 2 /*SHA1 hash algorithm*/

TM_IKE_RIPEMD 4 /*RIPEMD hash algorithm*/

Tiger Hash algorithm is not supported by Treck IKE.

1.3.1.4 AUTH METHOD

TM_IKE_PRESHARED_KEY 1 /* preshared key */

TM_IKE_DSS_SIG 2 /* DSA signature */

TM_IKE_RSA_SIG 3 /* RSA signature */

1.3.1.5 DH GROUP DESCRIPTION

TM_DHGROUP_1 1 /*768 MODP group*/

TM_DHGROUP_2 2 /* 1024 MODP group */

TM_DHGROUP_5 5 /* 1536 MODP group */

We support the predefined Diffie-Hellman MODP group 1, 2 and 5. Treck IKE only supports predefined MODP groups. Group 3, group 4 and custom groups are not supported.

1.3.1.6 LIFE TYPE

TM_IKE_LIFETYPE_SECONDS 1 /* life time in seconds */

TM_IKE_LIFETYPE_KBYTES 2 /* life time in kilo-bytes */

1.3.1.7 LIFE DURATION

Treck IKE requires that the SA life duration must be either two octets (0 to 18.2 hours of lifetime for AF bit set case) or four octets (0 to 136 years, AF bit not set) long. No other data length is supported.

1.3.1.8 PSEUDO RANDOM FUNCTION

Treck IKE uses the HMAC version of the negotiated hash algorithm to be the PRF. No other PRF is supported.

1.3.1.9 KEY LENGTH

Some encryption algorithms may accept different size of key. For Treck IKE, the following key size range is supported. And as RFC recommended, Treck IKE doesn't support key length that is not of multiple of 8 Bits (1 octet byte). For those algorithms with fixed key length, no key-length attribute is accepted. (RFC 2409 prohibits such behavior. This attribute MUST NOT be used when the specified encryption algorithm uses a fixed length key.)

TM_IKE_DES_CBC	64 Bits
TM_IKE_BLOWFISH_CBC	40 to 448 Bits
TM_IKE_RC5_R16_B64_CBC	40 to 2040 Bits
TM_IKE_3DES_CBC	192 Bits
TM_IKE_CAST_CBC	40 to 128 Bits
TM_IKE_AES_CBC	128, 192, or 256 Bits
TM_IKE_TWOFISH_CBC	128, 192, or 256 Bits

1.3.2 Phase Two Negotiation Attributes

For phase 2, we support the following attributes, per RFC 2407.

1.3.2.1 PROTOCOL IDENTIFIER AND TRANSFORM IDENTIFIER

Treck IKE supports the following Phase 2 protocol Identifier.

TM_DOI_PROTO_IPSEC_AH	2
TM_DOI_PROTO_IPSEC_ESP	3

And protocol TM_DOI_PROTO_IPCOMP is not supported currently.

For TM_DOI_PROTO_IPSEC_AH, the following transform identifiers are supported:

TM_DOI_AH_MD5	2 /* HMAC_MD5_96 */
TM_DOI_AH_SHA	3 /* HMAC_SHA1_96 */
TM_DOI_AH_RIPEMD	5 /* HMAC_RIPEMD160_96, use this only in Treck products */

For TM_DOI_PROTO_IPSEC_ESP, the following transform identifiers are supported:

TM_DOI_ESP_DES_IV64	1
TM_DOI_ESP_DES	2
TM_DOI_ESP_3DES	3
TM_DOI_ESP_RC5	4 (need patent license to run)
TM_DOI_ESP_IDEA	5 (need patent license to run)
TM_DOI_ESP_CAST128	6
TM_DOI_ESP_BLOWFISH	7
TM_DOI_ESP_NULL	11
TM_DOI_ESP_AES	12

Treck IKE also supports TWOFISH. However, interoperation with other implementation is not guaranteed.

TM_DOI_ESP_TWOFISH	13
--------------------	----

1.3.2.2 LIFE TYPE

TM_DOI_SATOL_LIFE_SECONDS	1 /* life time in seconds */
TM_DOI_SATOL_LIFE_KBYTES	2 /* life time in kilo-bytes */

1.3.2.3 LIFE DURATION

Treck IKE requires that the SA life duration must be either 2 octets (0 to 18.2 hours of lifetime for AF bit set case) or 4 octets (0 to 136 years, AF bit not set) long. No other data length is supported.

1.3.2.4 DH GROUP DESCRIPTION

TM_DHGROUP_1	1 /*768 MODP group*/
TM_DHGROUP_2	2 /* 1024 MODP group */
TM_DHGROUP_5	5 /* 1536 MODP group */

We support the predefined Diffie-Hellman MODP group 1, 2 and 5. Treck IKE only supports predefined MODP groups. Group 3, group 4 and custom groups are not supported.

1.3.2.5 ENCAPSULATION MODE

TM_DOI_ENCAPMODE_TUNNEL	1
TM_DOI_ENCAPMODE_TRANSPORT	2

Transport mode and tunnel mode are both supported

1.3.2.6 AUTHENTICATION ALGORITHM

TM_DOI_AUTHALG_HMAC_MD5	1
TM_DOI_AUTHALG_HMAC_SHA_1	2
TM_DOI_AUTHALG_HMAC_RIPEMD	5 /* use this only in Treck products */

Treck IKE supports the above three authentication algorithms.

1.3.2.7 KEY LENGTH

Some encryption algorithms may accept a different size key. For Treck IKE, the following key size range is supported. As the RFCs recommended, Treck IKE does not support key lengths that are not a multiple of 8 Bits (1 octet byte). For those algorithms with a fixed key length, a no key-length attribute is accepted. (RFC 2409 prohibits such behavior. This attribute **MUST NOT** be used when the specified encryption algorithm uses a fixed key length.)

TM_DOI_ESP_DES_IV64	64 Bits
TM_DOI_ESP_DES	64 Bits
TM_DOI_ESP_3DES	192 Bits
TM_DOI_ESP_RC5	40 to 2040 Bits
TM_DOI_ESP_CAST	40 to 128 Bits
TM_DOI_ESP_BLOWFISH	40 to 448 Bits
TM_DOI_ESP_NULL	don't care
TM_DOI_ESP_AES	128, 192, or 256 bits

Treck IKE also supports TWOFISH transformer, but interoperation is not guaranteed.

TM_DOI_ESP_TWOFISH	128, 192, or 256 Bits
--------------------	-----------------------

1.3.2.8 KEY ROUNDS

For some encryption algorithms, a user defined ROUND value may be used. However, it affects the interoperation a lot. We always use the default round value. Like, RC5, Blowfish, Cast algorithm, we are always using 16 rounds.

1.3.3 Treck IKE features

Treck IKE provides the following features:

- Supports main mode, aggressive mode, and quick mode.
- Supports Pre-shared Key, DSA and RSA signatures authentication methods
- Supports Diffie-Hellman predefined group 1, 2 and 5.
- Support ID types - IPv4 and IPv6 Address, Subnet and Range, FQDN, USER_FQDN, and ASN1_DN
- Supports encryption algorithm DES, 3DES, BLOWFISH, CAST, AES (RIJNDAEL) and TWOFISH.
Note: User needs to get RC5-R16-B64 and IDEA patent license before requiring source code.
- Supports Perfect Forward Secrecy. If Treck IKE is the initiator for phase two, it always uses the phase one Diffie-Hellman group as the PFS proposal. If Treck IKE is the responder, it accepts group 1, 2 or 5.
- Supports Hash algorithms MD5, SHA1 and RIPEMD. The corresponding HMAC version hash algorithm is used as the Pseudo Random Function.
- Ignores commit bit and auth-only bit based on IKE bake-offs. Uses pre-setup incoming SA to avoid losing packet. Unique message ID list is maintained to work against replay.
- Cookie is generated using continuously refreshing secret key. Cookie verification is configurable.
- Lifetime is based on time and kilobytes.
- New group mode is not supported.
- Supports ISAKMP Informational Exchanges, including including DELETE, INITIAL-CONTACT message. Other informational messages will be silently discarded. (RFC states we MAY send an error notification.) We do not send RESPONDER_LIFETIME exchange because any participant may choose to renegotiate at its preferred time.
- One SA negotiation at each quick mode exchange
- For the pair exchanges, the requester is responsible for retransmit. The responder will retransmit only upon receiving a retransmitted message from the requester.
- Supports both IPv4 and IPv6.
- Packets are queued when negotiating SA, while time and bytes limitation apply.
- Once connection is established, Treck IKE will try to keep continuous channel between peers. Phase 1 ISAKMP SA will be automatically rekeyed once expires.

2 Security Policy Database (SPD) and Security Association Database (SAD)

SPD specifies the policies that determine the disposition of all IP traffic inbound or outbound from a host or a security gateway. SAD is a security association table, containing parameters that are associated with each security association.

2.1 SPD

The SPD must be consulted during the processing of all traffic (both inbound and outbound), including non-IPsec traffic. The policy entries in SPD are totally ordered, and the first matched policy will be used to process the traffic.

For example, we have two IPv4 subnet networks, subnet A 1.1.1.0/24 and subnet B 2.2.1.0/24, with security gateway GA and GB, as shown in Figure 5. This diagram could also be degraded to a host A to subnet B, or even host A to host B, in such cases, GA or GB becomes an IPsec-enabled host.

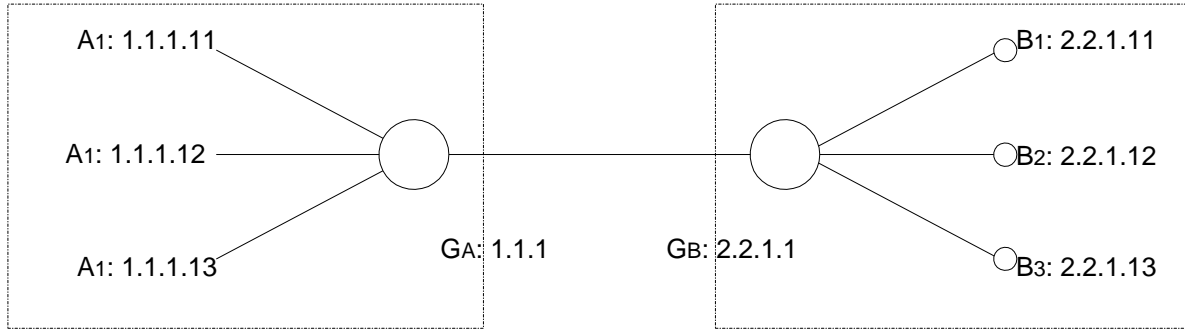


Figure 5 Subnet Network 1.1.1.0/24, and 2.2.1.0/24

The SPD table at GA is shown in Table 1.

Table 1 SPD table at G_A (Suppose we have 10 policy entries at G_A)

index	Direction	Local IP (sharing?)	Local port (sharing?)	Remt ip (sharing?)	Remt port (sharing?)	Protocol (sharing?)	In-bound 1 st SA entry	Out-bound 1 st SA entry	Action	Policy 1 st content
9	In & Out	1.1.1.12 Y	80 Y	2.2.1.0/24 Y	Any Y	TCP Y	sa15	sa25	IPSEC	Cont1
8	In & Out	1.1.1.0/24 Y	Any Y	2.2.1.0/24 Y	Any Y	Any N	sa10	sa20	IPSEC	Cont3
...	...									
...	In									
.	Out									
0	In & Out	Any	Any	Any	Any	Any	NULL	NULL	BY-PASS	NULL

Preference ↓

The policy order determines the packet processing behavior. The policy table is always looked up from the very top (the most preferred one) to the very bottom (the least preferred one, with policy index equal to zero). The first matched policy will be used to process the packet. The policy index number 0 is recommended to be an ALL-BYPASS or ALL-DISCARD policy, otherwise, if a packet cannot find a suitable policy, a 'NO-POLICY' error will return.

In table 1, policy index No.9 is the preferred policy applying to both INBOUND and OUTBOUND matched traffic. If traffic does not match policy No.9's selector, we check against the next policy, i.e. policy No.8, and then policy No. 7, and so on down to policy No. 0.

A policy has one or more policy contents, and each content corresponds to an IPsec protocol, either AH or ESP, but not both. If a policy needs both ESP and AH, two separate policy contents must be used, and they are linked together

through the next content pointer. For example, policy No.9 requires both ESP and AH. ESP tunnel mode must be performed before the AH transport mode protection. The policy contents are shown in figure 6.

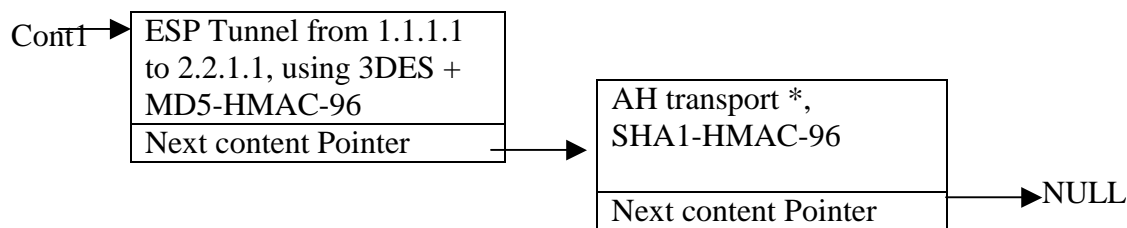


Figure 6 Policy Content Linked List

* For data format like IP + AH + ESP + IP+ DATA, Treck calls the ESP in tunnel mode and the AH in transport mode, and for data format like IP + AH + IP + ESP + IP+ DATA, Treck calls both ESP and AH in tunnel mode.

2.2 SAD

SA is generated according to the policy contents. Each SA is able to provide security service of either AH or ESP, but not both. If both ESP and AH are required, a group of SA (i.e. SA bundle) is required.

The YES-NO sharing flag of each policy determines how to build SA according to the policy. It indicates whether the corresponding selector field uses policy value or packet value. (See RFC 2401 Page 15 for details of policy value against packet value) If NO sharing is defined, we use packet value to build SA. If YES is defined, we use policy value to build SA. For example, policy index No.8, the policy selector/protocol field is set to be ANY but prohibits sharing (NO sharing), thus, although both TCP and UDP matches this policy, they must use different SA, i.e., use different encryption key and authentication keys to protect the traffic. If sharing is allowed, we will use the policy value (ANY protocol) to build SA, and both TCP and UDP will use the same SA.

The SAD table is a hash table. The triple value <SPI, destination IP, and Protocol> are hashed. Each SA also contains traffic selector, which may or may not equal to the policy traffic selector. (When SA sharing is fully granted, these two selectors will be the same).

2.3 Outbound Processing

If IPsec is desired, each outbound packet is compared against the SPD to determine what processing is required for the packet.

- 1. Match the packet's selector fields against the outbound policies in the SPD to locate the FIRST appropriate policy.
- 2. Determine the action according to the policy located. If discard is required, the packet is discarded, (packet information may be logged). If packet is allowed to bypass IPsec processing, skip the following IPsec processing. Go to step 6.
- 3. If IPsec processing is required, locate the appropriate SA generated by this policy's innermost policy content using the first-outbound-sa pointer for a certain policy. If no SA is found and IKE is not activated, drop the packet. Otherwise, if IKE is activated, queue the packet and initiate IKE negotiation.
- 4. Use the SA found in last step to do the required IPsec processing.
- 5. Go back to step 3 to determine if there is outer policy content. Find the SA according to the outer policy content if outer policy content exists. If no more outer policy content, go to step 6.
- 6. Send the packet out.

At step 3 above, the packet will be queued if IKE starts. The packet processing procedure will be resumed when IKE finishes the negotiation and the SA is ready.

For TCP socket, the SA(s) used to protect the traffic will be queued in the socket pointer. The next time sending packets through this socket, policy and SA lookup is not required.

2.4 Inbound Processing

The following steps are taken to process IPsec inbound packet [see RFC 2401]:

- 1. When an IPsec protected packet is received, use the packet's destination address (outer IP header), IPsec protocol, and SPI to look up the SA in the SAD. If the SA lookup fails, drop the packet and log/report the error.
- 2. Use the SA found in (1) to do the IPsec processing, e.g., authenticate and decrypt. Meanwhile, save the SA and its order for later verification usage.
- 3. Process the next header, if it is IPsec header, go back to step 1; if it is another IP header (SA found in (1) must be in tunnel mode), remove the tunnel, and process the next header again. If a transport protocol header is encountered, go to step 4
- 4. Find the incoming policy in the SPD using the inner IP header information.
- 5. Check whether the required IPsec processing has been applied. i.e., verify that the SA's saved in (2) match the kind and order of SA's required by the policy found in (4).

If the SA verification procedure returns no error, pass the resulting packet to the transport layer or forward the packet.

2.5 Treck SPD and SAD features

- SPD management APIs are provided with the capability of single policy adding, bundle policy adding, bulk-loading policy adding, policy deleting and SPD clearing.
- SAD management APIs are provided with the capability of adding, removing, and clearing.
- SPD and SAD are maintained in separate tables.
- SPD is searched linearly in order to find the first match, while SAD is maintained as hash table.
- SPD and SAD selectors are based on local and remote IP addresses, in host type, subnet type or range type. (Fully qualified domain name is also supported at the user's interface), local and remote ports, and traffic protocol. SA using policy value or packet value is also configurable.

3 Settings in trsystem.h

All user accessible IPsec and IKE macros are included in the header file trsystem.h, and all public API prototypes are included in trsecapi.h. To support IPv6, the user must define TM_USE_IPV6.

3.1 TM_USE_IPSEC

This macro must be defined to have IPsec behavior.

3.2 TM_USE_IKE

TM_USE_IKE must be defined to use IKE. If manual keying is desired, do not define this macro, and you must manually add all necessary SA's. Define TM_USE_IKE to let IKE module negotiate SA and manage SA Database.

3.3 TM_IPSEC_DEBUG

Define TM_IPSEC_DEBUG to debug IPsec authentication/encryption.

3.4 TM_IKE_DEBUG

Define TM_IKE_DEBUG to debug IKE.

3.5 TM_IPSEC_USE_ANTIPLAY

Define TM_IPSEC_USE_ANTIPLAY to check the sequence number. If this macro is defined and sequence number overflows, a new SA must be negotiated. Otherwise, if this macro is not defined, sequence number will be reused if overflows. Even this macro is defined, Treck IPsec does not perform anti-replay checking for non-authenticated ESP packet (per RFC 2406). Also we suggest not to define this macro if TM_USE_IKE is not defined.

3.6 TM_IPSEC_INCOMING_ICMP_BYPASS

Define this macro to bypass incoming ICMP policy checking. If this macro is not defined, all ICMP incoming packets will be checked against local IPsec policy. Unprotected ICMP packets, like ping messages, will be dropped if they do not match the local policy requirement. And note that many routers will not be IPsec-capable for some time to come, for the above reasons, the default status of this macro is defined.

3.7 TM_IPSEC_INCOMING_ICMP_NOSRCCHK

If we bypass the ICMP policy checking (by defining TM_IPSEC_INCOMING_ICMP_BYPASS), it does not matter if this macro is defined or not. While if we do need checking incoming ICMP message against the local policy, we need handle situation that a router is tunneling ICMP packets generated by other routers. The routers at the tunnel source and destination would have established a tunnel mode SA for their communication. This SA would be used to forward ICMP error messages. However, the source address of the inner header (other router) will not be the tunnel source address. The policy checking at the destination will fail. Define this macro to bypass ICMP source address checking.

3.8 TM_6_IPSEC_ICMPV6_NDMLD_BYPASS

If you want to bypass IPsec protection to avoid the chicken-and-egg problem for ICMPv6 ND and MLD message, please define this macro. If TM_USE_IKE is defined, you should define this macro, otherwise, you have to manually add the corresponding SA as the first thing.

3.9 TM_IPSEC_DEFAULT_REPLAY_SIZE

Defines the replay window size. The SA sequence number will be checked using this window size. Default value is 32. If it is set to zero, no replay checking will be performed.

3.10 TM_IKE_DHSECRET_DEFAULT_SIZE

Defines the Diffie-Hellman local secret size in bytes. NIST implementation suggests this value to be 32 bytes.

3.11 TM_IKE_PFS_KEY_ENABLED

Define this macro if the Perfect Forward Secrecy feature is desired. The PFS has two options, PFS of KEY, and PFS of Identity. This macro applies to PFS of KEY.

3.12 TM_IKE_PHASE1_AGGRESSIVE_ENABLED

For IKE main mode negotiation, if the Treck IKE is running as the initiator, there are two choices: use Main mode, or use Aggressive mode. Define this macro to use Aggressive mode. If Treck IKE is running as the responder, this macro has no effect because we accept both Main mode and Aggressive mode. If the Treck stack initiates an Aggressive mode negotiation but receives TM_ISAKMP_INVALID_EXCHANGE_TYPE notification message or encounters a LAST_TIME_OUT event, all Aggressive mode states will be cleared, and Treck IKE will initiate a Main mode negotiation instead.

3.13 TM_IKE_AGGRESSIVE_DHGROUP

For aggressive mode, we must know which Diffie-Hellman group we are going to use. (in order to send proper KE payload) Default value for this macro is TM_DHGROUP_2, i.e., use Diffie-Hellman group 2 in aggressive mode.

3.14 TM_IKE_INITIAL_CONTACT_ENABLED

Treck IKE will try to maintain a continuous channel with the IKE peer. Absence of phase one channel means that this is a new connection request. Define this macro to notify the peer. All previously setup-ed SA's must be deleted. We only receive INITIAL_CONTACT message in protected mode. INITIAL_CONTACT in clear mode will be discarded. In Aggressive mode, we do not send an INITIAL_CONTACT message as the RFC recommends

3.15 TM_IKE_USE_DEFAULT_PRESHARED_KEY

Treck IKE will try to find the pre-shared key for specified identification, if no pre-shared key found, and if TM_IKE_USE_DEFAULT_PRESHARED_KEY is defined, Treck IKE will use the default pre-shared key to negotiate. If it is not defined and no pre-shared key is found, error will be returned.

3.16 TM_IPSEC_BYPASS_NESTIKE_PACKET

Define this macro to bypass any nested IKE packet. If this macro is defined, Treck IKE will not do any protection for nested IKE packet. Otherwise, nested IKE packet are subject to IPsec policy check. This macro is not defined by default.

3.17 TM_IPSEC_USE_SW_CRYPTENGINE

Define TM_IPSEC_USE_SW_CRYPTENGINE if any TRECK software implementation of hashing algorithms, encryption/decryption algorithms, or public key exchange algorithms is used. If this macro is not defined, any of these algorithms must be provided by other vendor's software implementation or hardware implementation.

3.18 TM_IPSEC_USE_HF7951_CRYPTENGINE

Define TM_IPSEC_USE_HF7951_CRYPTENGINE if IPsec hardware accelerator HIFN 7951 is used.

3.19 TM_USE_PKI

Define this macro in order to use DSA or RSA digital signatures authentication in IKE Phase 1. Treck PKI currently includes X509v3 certificate (ASN1, PEM and DER format), and RSA/DSA signature and verification.

3.20 TM_PKI_CHECK_CERT_ALIVE

Define this macro if PKI checks if a certificate is alive before loading the certificate. TM_PKI_TIME must be also defined as the PKI start time if the TM_PKI_CHECK_CERT_ALIVE is defined, the default value of TM_PKI_TIME is "03051400000Z" that means 2003, May.14 and Z means Zulu, or Greenwich Mean Time.

3.21 TM_PKI_CERT_NOT_VERIFY

Define this macro if PKI does NOT want to verify a certificate while loading it.

3.22 TM_USE_RIPEMD

Define this macro to use RIPEMD160-96 hash algorithm

3.23 TM_USE_DES

Define this macro to use DES algorithm

3.24 TM_USE_3DES

Define this macro to use 3DES algorithm

3.25 TM_USE_BLOWFISH

Define this macro to use BLOWFISH for ESP transform.

3.26 TM_USE_RC5

Define this macro to use RC5 for ESP transform. Must have patent license to run.

3.27 TM_USE_CAST128

Define this macro to use CAST128 for ESP transform

3.28 TM_USE_AES

Define this macro to use AES (RIJNDAEL) for ESP transform

3.29 TM_USE_TWOFISH

Define this macro to use TWOFISH for ESP transform

3.30 TM_PUBKEY_USE_DIFFIEHELLMAN

Define this macro to use software implementation of Diffie-Hellman module, group 1, 2, and 5 are supported

3.31 TM_PUBKEY_USE_RSA

Define this macro to use RSA module. User must also define TM_USE_PKI in order to use RSA.

3.32 TM_PUBKEY_USE_DSA

Define this macro to use DSA module. User must also define TM_USE_PKI in order to use DSA.

4 Function Reference

4.1 IPsec and IKE initialization

IPsec and IKE initialization procedure allocates memory for the IPsec and IKE global variables. To use IPsec, you must define `TM_USE_IPSEC` macro in `trsystem.h`. To use IKE, you must define `TM_USE_IKE` macro in `trsystem.h`.

4.1.1 `tfUseIpsec`

```
int                tfUseIpsec
(
void
);
```

Description

`tfUseIpsec` must be called after `tfStartTreck()`, and before `tfStartIke` (if IKE is defined). We suggest calling it before `tfOpenInterface`. `tfUseIpsec` allocates and initializes the IPsec global variable `tvIpsecPtr`.

Parameters

None

Returns

Value	Meaning
<code>TM_ENOERROR</code>	success
<code>TM_EIPSECNOTINITIALIZED</code>	<code>TM_USE_IPSEC</code> is not defined
<code>TM_EALREADY</code>	IPsec global variable <code>tvIpsecPtr</code> is not NULL before this call
<code>TM_ENOBUFS</code>	No more buffers for <code>tvIpsecPtr</code>

Example

```
{
...
/* start treck */
    errorCode = tfStartTreck();
    if (errorCode != TM_ENOERROR)
    {
    }
...

/* Initialize IPsec global variables */
#ifdef TM_USE_IPSEC
    errorCode = tfUseIpsec();
    if (errorCode != TM_ENOERROR)
    {
```

```
    }  
#endif /* TM_USE_IPSEC*/  
  
/* Open the added Loop back interface */  
    errorCode = tfOpenInterface( ...);  
    ...  
}
```

4.1.2 tfStartIke

```
int                tfStartIke
(
  unsigned int     ipaddrType,
  unsigned int     ipaddrLength,
  ttUserVoidPtr   ipaddrPtr
);
```

Description

If IKE is required, either **tfStartIke** or **tfStartEnhancedIke** must be called after **tfOpenInterface** (or **tfNgOpenInterface for IPv6**). If the ID in IKE phase 1 exchange is the same as the IP address we run IKE, **tfStartIke** is used. Otherwise, if ID is anything other than the IP address we run IKE, **tfStartEnhancedIke** should be used. **tfStartIke** gets the identification (the IP address) of IKE participant, allocates and initializes the IKE global variable *tvIkePtr*, opens an IKE socket with port 500, registers callback function(internal) to the IKE socket for receiving event, and starts refreshing the cookie secret pool regularly.

Parameters

Value	Meaning
<i>ipaddrType</i>	Identification IP address type of Treck IKE participant. It could be: TM_DOI_ID_IPV4_ADDR, a single four octets IPv4 address, or TM_DOI_ID_IPV6_ADDR, a single 16 octets IPv6 address.
<i>ipaddrLength</i>	The length of identification For TM_DOI_ID_IPV4_ADDR, it should be 4, For TM_DOI_ID_IPV6_ADDR it should be 16,
<i>ipaddrPtr</i>	Pointer to the IKE participant identification. If idType = TM_DOI_ID_IPV4_ADDR, it points to the four octet IPv4 address (in network order) If idType = TM_DOI_ID_IPV6_ADDR, it points to the sixteen octet IPv6 address (in network order)

Returns

Value	Meaning
TM_ENOERROR	success
TM_EIPSECNOTINITIALIZED	TM_USE_IKE is not defined
TM_EALREADY	IKE global variable tvIkePtr is not NULL before this call
TM_ENOBUFS	No more buffers available
Other error	socket error, user tfStrError(error code) to get the detail error information

Example

This example starts IKE using identification type `TM_DOI_ID_IPV4_ADDR` "1.1.1.1"

```

{
...
    ttUserIpAddress  clientIpAddr = inet_addr("1.1.1.1");
    ttUserIpAddress  mask = inet_addr("255.255.255.0");

/* start treck */
    errorCode = tfStartTreck();
    if (errorCode != TM_ENOERROR)
    {
    }
...
/* Initialize IPsec global variables */
#ifdef TM_USE_IPSEC
    errorCode = tfUseIpsec();
    if (errorCode != TM_ENOERROR)
    {
    }
#endif /* TM_USE_IPSEC*/

/* Open the added Loop back interface */
    errorCode = tfOpenInterface( ...);

#ifdef TM_USE_IKE
    errorCode = tfStartIke(TM_DOI_ID_IPV4_ADDR,
                            4,
                            (void*)&clientIpAddr);
    if (errorCode != TM_ENOERROR)
    {
    }
#endif /* TM_USE_IKE*/
...
}

```

4.1.3 tfStartEnhancedIke

```

int                                tfStartEnhancedIke
(
unsigned int                        idType,
unsigned int                        idLength,
ttUserVoidPtr                      idPtr,
const struct sockaddr_storage TM_FAR *
                                   ikeSockaddrPtr)
);

```

Description

If the ID in IKE phase 1 exchange is different than the IP address we run IKE, for example, we use domain name, email address, username etc, **tfStartEnhancedIke** should be used. Actually, **tfStartEnhancedIke** can do anything that **tfStartIke** can do. **tfStartIke** is just a special case of **tfStartEnhancedIke**.

Parameters

Value	Meaning
<i>idType</i>	Identification type of Treck IKE participant. Could be: TM_DOI_ID_IPV4_ADDR, a single four octets IPv4 address TM_DOI_ID_FQDN, a fully qualified domain name string, like "foo.bar.com". TM_DOI_ID_USER_FQDN, a fully qualified username string, like jok@foo.bar.com TM_DOI_ID_DER_ASN1_DN, a Distinguished name in the host certificate TM_DOI_ID_IPV6_ADDR, a single 16 octets IPv6 address. Please note that, if idType is IPv4 or IPv6 address, you may want to use tfStartIke .
<i>idLength</i>	The length of identification For TM_DOI_ID_IPV4_ADDR, it should be 4, For TM_DOI_ID_IPV6_ADDR it should be 16, For TM_DOI_ID_DER_ASN1_DN, it should be zero. For other idType, it will be the length of name string.
<i>idPtr</i>	Pointer to the IKE participant identification. If idType = TM_DOI_ID_IPV4_ADDR, it points to the four octet IPv4 address (in network order) If idType = TM_DOI_ID_IPV6_ADDR, it points to the sixteen octet IPv6 address (in network order) If idType = TM_DOI_ID_FQDN or TM_DOI_ID_USER_FQDN, it points to the name string.
<i>ikeSockAddrPtr</i>	Pointer to a struct sockaddr_storage. It has the IP address information of the IKE participant.

Returns

Value	Meaning
TM_ENOERROR	success
TM_EIPSECNOTINITIALIZED	TM_USE_IKE is not defined
TM_EALREADY	IKE global variable tvIkePtr is not NULL before this call
TM_ENOBUFS	No more buffers available
Other error	socket error, user tfStrError (error code) to get the detail error information

Example

This example starts IKE using identification type TM_DOI_ID_FQDN for digital signature authentication, and is using IP address "3FFF::1"

```
{
...
    struct sockaddr_storage  clientIpAddr;
...
    tm_bzero(&clientIpAddr, sizeof(struct sockaddr_storage));
    clientIpAddr.ss_len = sizeof(struct sockaddr_in6);
    clientIpAddr.ss_family = PF_INET6;
    inet_pton(PF_INET6, "3FFF::1",
              &clientIpAddr.addr.ipv6.sin6_addr);
/* assert the return value of inet_pton is valid*/
...
/* start treck */
    errorCode = tfStartTreck();
/* assert the return value */
...
/* Initialize IPsec global variables */
#ifdef TM_USE_IPSEC
    errorCode = tfUseIpsec();
    if (errorCode != TM_ENOERROR)
    {
    }
#endif /* TM_USE_IPSEC*/
/* Open the interface */
    errorCode=tfNgOpenInterface(...);
/* assert errorCode */
...
#ifdef TM_USE_IKE
    errorCode = tfStartEnhancedIke(TM_DOI_ID_ FQDN ,
                                  14,
                                  (void*)"www.treck.com",
                                  &clientIpAddr);
/* assert return value */
#endif /* TM_USE_IKE*/
...
}
```

4.1.4 tfIpsecUninitialize

```
int                tfIpsecUninitialize(void);
```

Description

This function will uninitialize IPsec and IKE. It will clear all the policy entries as well as all IPsec SAs, and it will close the IKE socket(s), and free the raw buffer allocated for tvIpsecPtr and tvIkePtr. After this call, the stack will run as if tfUseIpsec has not been called.

Parameters

Value	Meaning
<i>none</i>	

Returns

Value	Meaning
TM_ENOERROR	success
Other Value	Uninitialize failed

4.1.5 tfIpsecSetOptions

```
int          tfIpsecSetOptions
(
ttUser32Bit option,
ttUser8Bit  value
);
```

Description

This function is used to change Treck IPsec/IKE behavior during the running time. Most of these behaviors can be achieved by defining corresponding macros. `tfIpsecSetOptions` provides a convenient way to change the IPsec/IKE behavior without having to recompile the whole stack. This function must be called after `tfUseIpsec()` call.

Parameters

Value

option

Meaning

Identifies the option user wants to change.

Valid options are:

`TM_IPSECOPT_ANTIREPLAY`, determines anti-replay is used or not.

Value 1 will enable anti-replay, value 0 disable it.

`TM_IPSECOPT_PFSKEY`, determines if Perfect Forward Secrecy is used of not.

value 1 will enable PFS, value 0 disables it.

`TM_IPSECOPT_AGGRESS`, determines if aggressive mode is used in IKE phase 1.

Value 1 will enable it, value 0 disable it.

`TM_IPSECOPT_AGGRESSDHGROUP`, determines which Diffie-Hellman group is going to be used in aggressive mode. Value must be 1, 2 or 5.

`TM_IPSECOPT_INITCONTACT`, determines if Initial Contact message be sent to the peer or not. Value 1 will enable IKE to send Initial contact upon finishing IKE phase 1 negotiation. Value 0 will disable it.

`TM_IPSECOPT_PKICERTCHECKALIVE`, determines if certificate lifetime is to be checked or not. Value 1 will enable checking. Value 0 will disable it.

`TM_IPSECOPT_PKICERTNONVERIFY`, determines if certificate is going to be verified or not. Value 1 will NOT verify certificate.

`TM_IPSECOPT_ICMPBYPASS`, determines if we bypass all ICMP packet or not. Value 1 will bypass all ICMP packet, no IPsec will be applied or check against.

`TM_IPSECOPT_ICMPSRCCHKBYPASS`, determines if ICMP source address be checked against IPsec policy or not. Value 1 will bypass this check.

`TM_IPSECOPT_ICMP6NDMLDBYPASS`, determines if ICMPv6 bypass IPsec policy or not. Value 1 will bypass IPsec.

`TM_IPSECOPT_NESTIKE_BYPASS`, determines if nested IKE packet be checked against IKE or not. Value 1 will bypass all IKE packet. Value 0 will force all IKE packets go through IPsec policy checking.

`TM_IPSECOPT_DFBIT`, determines how to set the outside IP header's Dont Fragment bit if tunnel is used. Valid value can be `TM_IPSEC_DFBIT_CLEAR`, `TM_IPSEC_DFBIT_SET` or `TM_IPSEC_DFBIT_COPY`.

value

Set the option value. In most cases, a non-zero value will enable the option, a zero value disables the option. For option like

`TM_IPSECOPT_AGGRESSDHGROUP`, value can be 1, 2 or 5, which corresponds to DH group 1, 2 and 5 respectively.

Returns

Value	Meaning
TM_ENOERROR	success
-1	tfIpsecSetOptions call failed

Example

This example sets some IPsec options. Please note that tfUseIpsec is called before this function call.

```
{  
...  
    errorCode = tfUseIpsec();  
  
/* Set IPsec options AFTER you call tfUseIpsec */  
  
/* don't use Perfect Forward Secrecy feature */  
    tfIpsecSetOptions(TM_IPSECOPT_PFSKEY, 0);  
/* use aggressive mode to negotiate IKE phase 1 */  
    tfIpsecSetOptions(TM_IPSECOPT_AGGRESS, 1);  
/* aggressive mode use dh group #2 */  
    tfIpsecSetOptions(TM_IPSECOPT_AGGRESSDHGROUP, 2);  
/* don't send initial contact. */  
    tfIpsecSetOptions(TM_IPSECOPT_INITCONTACT, 1);  
...  
}
```

4.2 Pre-shared Key Management API

4.2.1 tfPresharedKeyAdd

```
int          tfPresharedKeyAdd
(
char TM_FAR * idPtr,
char TM_FAR * keyDataPtr,
int          options
);
```

Description

This function adds an entry to the pre-shared key database. The user must define TM_USE_IKE to use this function.

Parameters

Value	Meaning
<i>idPtr</i>	Pointer to the Identity of the IKE peer, with which we are going to use the pre-shared key. It could be a valid null-terminated IPv4, IPv6 address string, or fully qualified domain name string or username string.
<i>keyDataPtr</i>	Pointer to the pre-shared key null-ended string. Although pre-shared key is not necessarily ASCII string, there is no obvious disadvantage to use ASCII string to be pre-shared key.
<i>options</i>	Not used at present

Returns

Value	Meaning
TM_ENOERROR	success
TM_ENOBUFS	no memory available

Example

This example adds a pre-shared key entry for peer "2.2.2.1", so that whenever we negotiate SA's with peer "2.2.2.1", we are going to use pre-shared key "GasdfHOPS99adsad323\$\$\$"

```
{
    ...

    errorCode = tfPresharedKeyAdd("2.2.2.1",
                                  "GasdfHOPS99adsad323$$$",
                                  0);

    ...
}
```

4.2.2 tfPresharedKeyDelete

```
int                tfPresharedKeyDelete
(
char TM_FAR       * idPtr,
int               options
);
```

Description

Deletes the pre-shared key for peer *idPtr*. After the deleting, it is no longer able to negotiate with peer *idPtr* using pre-shared key authentication.

Parameters

Value	Meaning
<i>idPtr</i>	Pointer to the identity of the IKE peer, with which we are going to use the pre-shared key. The <i>idPtr</i> should be string format of an IPv4 address, IPv6 address, a domain name or a username.

Returns

Value	Meaning
TM_ENOERROR	Success
-1	The Identification showed in <i>idPtr</i> is not found in the pre-shared key database

Example

This example deletes the pre-shared key for peer "2.2.2.1"

```
{
    ...

    errorCode = tfPresharedKeyDelete ("2.2.2.1", 0);

    ...
}
```

4.2.3 tfPresharedKeyClear

```
void          tfPresharedKeyClear
(
int          options
);
```

Description

Deletes all the pre-shared keys. If our authentication method is pre-shared key, we are no longer available to negotiate with any peer, unless new pre-shared keys are added.

Parameters

Value	Meaning
<i>options</i>	Not used at present

Returns

No returns

Returns

Value	Meaning
TM_ENOERROR	Success
TM_EINVAL	Invalidate parameter or certificate format

Example

This example adds a root CA certificate with PEM format and a local certificate that is stored in local disk with PEM format

```
{
    static char tlcACert[] =
        "MIIEPDCCAYQCAQIwDQYJKoZIhvcNAQEEBIAAwDELMAkGA1UEBgQCVMx CzA JBgNV\
        ...
jLtdUV1B69/1IeukzWnKeuAeN+jju2nioOEeFGiuyCg=" ;
    ...
    errorCode = tfPkiCertificateAdd (
        tlcACert,
        TM_PKI_CERTIFICATE_PEM|TM_PKI_CERTIFICATE_STRING,
        TM_PKI_CERT_NONLOCAL,
        TM_PKI_ROOT_CAID,
        tm_strlen(TM_PKI_ROOT_CAID),
        0 );
    ...
    errorCode = tfPkiCertificateAdd(
        "c:/cert/duser.crt",
        TM_PKI_CERTIFICATE_PEM,
        TM_PKI_CERT_LOCAL,
```

4.3 PKI API

4.3.1 tfUsePki

```
int          tfUsePki
(
void
);
```

Description

This function must be called before you try to add/delete certificate. The user must define `TM_USE_PKI` to use this function.

Parameters

Value	Meaning
No parameters.	

Return Value:

Value	Meaning
TM_ENOERROR	Successful return
TM_ENOBUFS	No buffer available
TM_EALREADY	tvPkiPtr is already allocated

4.3.2 tfPkiUninitialize

```
void          tfPkiUninitialize
(
void
);
```

Description

This function is called to uninitialize PKI. It will free all certificates and the PKI global variable tvPkiPtr.

Parameters

Value	Meaning
No parameters.	

Return Value:

Value	Meaning
No return	

4.3.3 tfPkiCertificateAdd

```
int          tfPkiCertificateAdd
(
ttUserVoidPtr  fileNamePtr,
int           fileFormat,
int           certType,
ttUserVoidPtr  idPtr,
int           idLength,
ttUserVoidPtr  caIdPtr
);
```

Description

This function adds an entry to the certificate database. The user must define `TM_USE_PKI` to use this function. If a certificate need to be verified (do NOT define `TM_PKI_CERT_NOT_VERIFY`), its CA certificate(s) must be added before the certificate is added. In addition, `tfPkiOwnKeyPairAdd` call to add a key pair for a local certificate must be called before adding the local certificate.

Parameters

Value	Meaning
<i>fileNamePtr</i>	Pointer to a certificate file name with directory information or PEM string of the certificate according following <i>fileFormat</i> value. Treck PKI right now supports PEM or DER certificate file format, and must have Treck FTP file system that supports DOS, Linux and ram files.
<i>fileFormat</i>	Specify the type and format in <i>fileNamePtr</i> with following macros. /* certificate with PEM (privacy enhanced mail) format */ TM_PKI_CERTIFICATE_PEM (tt8Bit)0x01 /* certificate with DER encoding format */ TM_PKI_CERTIFICATE_DER (tt8Bit)0x02 /* the above <i>fileNamePtr</i> is already a certificate string with PEM format, instead * of a file name. In this case, <i>fileNamePtr</i> must be PEM format string */ TM_PKI_CERTIFICATE_STRING (tt8Bit)0x04
<i>certType</i>	The certificate holder type, where /* the certificate is local */ TM_PKI_CERT_LOCAL 0 /* the certificate is general one, e.g., CA (certificate authority) or peer certificate */ TM_PKI_CERT_NONLOCAL 1 /* it is CRL (certificate Revocation list) */ TM_PKI_CERT_CRL 2
<i>idPtr</i>	Pointer to the Identification. If the certificate is ROOT CA, <i>idPtr</i> should be <code>TM_PKI_ROOT_CAID</code> .
<i>idLength</i>	Length of the Identification. <i>idLength</i> should be zero if the <i>idType</i> is <code>TM_DOI_ID_DER_ASN1_DN</code>
<i>caIdPtr</i>	Pointer to the Identification of CA. If it is 0, then Treck PKI will automatically search existing certificate database to find CA by matching DN (distinguished name). For <code>TM_PKI_ROOT_CAID</code> , <i>caIdPtr</i> should be 0.

```

        (void*)&myIpAddr.addr.ipv6.sin6_addr,
        sizeof(struct in6_addr),

        TM_PKI_ROOT_CAID);
}

```

In file c:/cert/duser.crt obtained from OpenSSL:

Certificate:

Data:

```

Version: 1 (0x0)
Serial Number: 3 (0x3)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=US, ST=CA, L=Riverside, O=Treck, OU=IPsec, CN=CA/

```

Email=ca@treck.com

Validity

Not Before: Jul 30 23:09:13 2002 GMT

Not After : Jul 30 23:09:13 2003 GMT

Subject: C=US, ST=CA, L=Riverside, O=Treck, OU=IPsec, CN=shang/

Email=abc@treck.com

Subject Public Key Info:

Public Key Algorithm: dsaEncryption

DSA Public Key:

pub:

6c:22:b7:bd:18:fd:7e:9f:60:7d:18:cc:63:73:6c:

a4:bd:52:00:d2:69:36:0e:61:01:91:8a:1a:de:d7

...

Signature Algorithm: md5WithRSAEncryption

86:fc:e9:f8:d1:69:60:f8:94:51:90:56:a9:f4:a8:19:eb:57:

65:82:34:c8:64:2f:8a:28:b6:1a:a9:48:58:1a:cc:54:ed:95:

...

—BEGIN CERTIFICATE—

```

MIIFgDCCBGgCAQMwDQYJKoZIhvcNAQEEBQAweDELMAkGA1UEBhMCVVMxCzAJBgNV
BAgTAkNBMRIWEAYDVQQHEw1SaXZlcnNpZGUxZjAMBgNVBAoTBUVVsbWljMQ4wDAYD

```

...

```

dOpsVGhohE5aH8n/7aPSPvtZeuCK6wKeiNaACY81T01Jre77MOLH4rh7KO33qbgU
Z4ZakU8095Vo+dTu4bH+MdMupWo=

```

—END CERTIFICATE—

4.3.4 tfPkiCertificateDelete

```
int          tfPkiCertificateDelete
(
ttUserVoidPtr  idPtr,
int            certType
);
```

Description

This function delete a certificate from the certificate atabase. The user must define `TM_USE_PKI` to use this function.

Parameters

Value	Meaning
<i>idPtr</i>	Pointer to the Identification of the certificate to be deleted.
<i>certType</i>	The certificate holder type. refer to above tfPkiCertificateAdd

Returns

Value	Meaning
TM_ENOERROR	success
TM_EINVAL	invalidate identification or certificate type

Example

This example deletes a ROOT CA certificate

```
{
    ...
    errorCode = tfPkiCertificateDelete (
        TM_PKI_ROOT_CAID,
        TM_PKI_CERT_NONLOCAL);    ...
}
```

4.3.5 tfPkiCertificateClear

```
int          tfPkiCertificateClear
(
int          certType
);
```

Description

This function clear all certificate with the certType, such as all certificate or all CRL. The user must define TM_USE_PKI to use this function.

Parameters

Value	Meaning
<i>certType</i>	certificate holder type

Returns

Value	Meaning
TM_ENOERROR	success
TM_EINVAL	invalidate value

Example

This example deletes all certificates

```
{
    ...
    errorCode = tfPkiCertificateClear (TM_PKI_CERT_NONLOCAL);
    ...
}
```

4.3.6 tfPkiOwnKeyPairAdd

```
int          tfPkiOwnKeyPairAdd
(
ttUserVoidPtr  fileNamePtr,
int           fileFormat,
int           keyType
);
```

Description

This function adds a public-private key pair of local (owner) certificate. The user must define `TM_USE_PKI` to use this function.

Parameters

Value	Meaning
<i>fileNamePtr</i>	Pointer to a certificate file name with directory information or PEM string of the certificate.
<i>fileFormat</i>	Specify the type and format in <i>fileNamePtr</i> . The format is following OpenSSL format.
<i>keyType</i>	key pair type, as: TM_PKI_RSA_KEY 0 /* key pair is RSA key */ TM_PKI_DSA_KEY 1 /* key pair is DSA key */

Returns

Value	Meaning
TM_ENOERROR	success
TM_ENOBUFS	no more memory available
TM_EINVAL	invalidate parameter

Example

This example adds a DSA public-private pair key

```
{
    static char tlPeerCertKey[] =
        "MIIDPgIBAAKCAQEAgdlaJM2XcQPia7+BZiF0YGTgotxpwn1aHmc/i/LF4ZN1/Ojb\
        ...
        IaiI9PopyolU0/zPC6SPjxVn";
    ...
    errorCode = tfPkiOwnKeyPairAdd (
        tlPeerCertKey,

        TM_PKI_CERTIFICATE_PEM|TM_PKI_CERTIFICATE_STRING,
        TM_PKI_DSA_KEY);
    ...
}
```

4.3.7 tfPkiPubKeyAdd

```
int          tfPkiPubKeyAdd
(
ttUserVoidPtr  fileNamePtr,
int           fileFormat,
int           certType,
ttUserVoidPtr  idPtr,
int           idLength
);
```

Description

Read Public Key from file (or memory) for a specified ID. The user must define `TM_USE_PKI` to use this function.

Parameters

Value	Meaning
<i>fileNamePtr</i>	Pointer to a certificate file name with directory information or PEM string of the certificate.
<i>fileFormat</i>	Specify the type and format in <i>fileNamePtr</i> . The format is following OpenSSL format.
<i>certType</i>	The certificate holder type, where <i>/* the certificate is local */</i> TM_PKI_CERT_LOCAL 0 <i>/* the certificate is general one, e.g., CA (certificate authority) or peer certificate */</i> TM_PKI_CERT_NONLOCAL 1 <i>/* it is CRL (certificate Revocation list) */</i> TM_PKI_CERT_CRL 2
<i>idPtr</i>	Pointer to the Identification. If the certificate is ROOT CA, <i>idPtr</i> should be TM_PKI_ROOT_CAID.
<i>idLength</i>	Length of the Identification. <i>idLength</i> should be zero if the idType is TM_DOI_ID_DER_ASN1_DN

Returns

Value	Meaning
TM_ENOERROR	success
TM_ENOBUFS	no more memory available
TM_EINVAL	invalidate parameter

4.4 ISAKMP Policy Management API

ISAKMP Policy means the IKE phase 1 policy in this document. In order for different IKE implementations to communicate and then negotiate to each other, they must agree with each other regarding the phase 1 parameters, such as hash algorithm, encrypt algorithm, diffie-hellman group, authentication method and lifetime type and value. After they agree with these phase 1 policy parameters (ISAKMP policy), they can establish phase 1 SAs, i.e. ISAKMP SAs. All phase 2 negotiation (i.e. IPsec SA negotiation) must be protected by the ISAKMP SAs.

As an IKE responder, TRECK IKE accepts any combination of supported algorithms and parameters. Such as MD5+DES+DHGroup 2, SHA1+3DES + DHGROUP5 or MD5+AEE+DHGROUP2. As an IKE initiator, TRECK IKE is able to send up to two transforms user configured using API in this section. If user didn't configure any transform, IKE will use the following default transform as its proposal.

```
{
  TM_IKE_3DES_CBC, TM_IKE_SHA1, TM_IKE_PRESHARED_KEY,
  TM_DHGROUP_2, 86400 /* lifetime in seconds */, 32768 /* lifetime in kbytes */, 0
}
```

For export version, it uses DES + MD5 + Preshared_key + GROUP1 and the same lifetime value.

4.4.1 `tfIkeAddPhase1Transform`

```
int tfIkeAddPhase1Transform(int          index,
                             int          encryptAlg,
                             int          encryptKeyLength,
                             int          hashAlg,
                             int          authMethod,
                             int          dhgroup,
                             int          lifeKbytes,
                             int          lifeSeconds)
```

Description

tfIkeAddPhase1Transform adds a user preferred phase 1 transform to IKE's phase 1 proposal. User is able to configure up to two transforms, the most preferred has index 0, the less preferred has index 1. If user configures any of them, the default transform won't be used, otherwise, if user fails to configure any of these transforms, the default transform will be used in the phase 1 proposal.

Parameters

Value	Meaning
<i>index</i>	Index of the transform. Can be either 0 or 1.
<i>encryptAlg</i>	The encryption algorithm to be used. The following values are supported (user must define the corresponding <code>TM_USE_XXX</code> macro, see section 3 Settings in <code>trsystem.h</code>) <code>TM_IKE_DES_CBC</code> <code>TM_IKE_BLOWFISH_CBC</code> <code>TM_IKE_3DES_CBC</code> <code>TM_IKE_CAST_CBC</code> <code>TM_IKE_AES_CBC</code> <code>TM_IKE_TWOFISH_CBC</code>
<i>encryptKeyLength</i>	Key length of the encryption algorithm. 0 means the default value. See section 5.1.2 <code>ttAhAlgorithm</code> and 5.1.3 <code>ttEspAlgorithm</code> for key length list.
<i>hashAlg</i>	The hash algorithm to be used. The following values are supported (see section 3 for usage of macro <code>TM_USE_RIPEMD</code>)

	TM_IKE_MD5
	TM_IKE_SHA1
	TM_IKE_RIPEMD
<i>authMethod</i>	The authentication method, see section 3 Settings in trsystem.h for macro TM_USE_PKI. You must define TM_USE_PKI to use digital signatures
	TM_IKE_PRESHARED_KEY /* pre-shared key*/
	TM_IKE_DSS_SIG /* DSS signature */
	TM_IKE_RSA_SIG /* RSA signature */
<i>dhGroup</i>	The Diffie-Hellman group to use. Valid values are: TM_DHGROUP_1 TM_DHGROUP_2 TM_DHGROUP_5
<i>lifetimeKbytes</i>	Lifetime in kilo-bytes. 0 means to use default value 32768. Valid value is between 10~2097152
<i>lifetimeSeconds</i>	Lifetime in seconds. 0 means to use default value 86400. Valid value is between 240 to 63072000

Returns

Value	Meaning
TM_ENOERROR	success
TM_EINVAL	Invalid value was used

4.4.2 tfIkeDeletePhase1Transform

```
int tfIkeDeletePhase1Transform(int index)
```

Description

tfIkeDeletePhase1Transform deletes a user configured phase 1 transform. If no user configured transform is found, the default transform will be used in the IKE phase 1 proposal.

Parameters

Value	Meaning
<i>index</i>	Index of the transform. Can be either 0 or 1.

Returns

Value	Meaning
TM_ENOERROR	success
TM_EINVAL	Invalid value was used

4.5 IPsec Policy Database Management API

If IPsec is used, all traffic is subject to policy check including incoming and outgoing, IPsec-protected, and non-IPsec-protected traffic. The policy must be there in order to receive or send packets (a simple policy example is that: For any source to any destination we BYPASS IPsec).

4.5.1 tfPolicyRestore

```
int                tfPolicyRestore
(
  ttIpsecPolicyPairPtr    pairListPtr,
  ttIpsecSelectorInStringPtr  sListPtr,
  ttPolicyContentInStringPtr  cListPtr,
  int                    npair
);
```

Description

tfPolicyRestore builds the SPD using bulk load method. For embedded systems, because we don't have file system, we have to load the policy using static data (rather than configuration files).

Parameters

Value	Meaning
<i>pairListPtr</i>	Pointer of type structure ttIpsecPolicyPair. For the structure reference, see section 5 Structure Reference.
<i>sListPtr</i>	Pointer of type ttIpsecSelectorInString
<i>cListPtr</i>	Pointer of type ttPolicyContentInString
<i>npair</i>	Number of pairs in the list pairListPtr.

Returns

Value	Meaning
TM_ENOERROR	success
Value other than TM_ENOERROR	error happens for policy restoring

Example

This example adds two policies at gateway 1.1.1.1:

From 1.1.1.0/24 to 2.2.2.1, any port, any protocol, ESP tunnel from 1.1.1.1 to 2.2.2.1, and then use AH transport. Using DES-CBC for encryption, and NULL authentication for ESP. HMAC_SHA1_96 algorithm for AH. Share the SA for all hosts inside subnet 1.1.1.0/24

From ANY to ANY, BYPASS

```
{
    int    errorCode;
    ttIpsecSelectorInString plcySelector[] =
    {
/* selector #0, ANY to ANY */
        {
/* local ip and mask, or ipmin and ipmax if selector is a range */
            (char*)0,          0,
/* remote ip and mask, or ipmin and ipmax if selector is a range */
            (char*)0,          0,
/* ip flags */
            (ttl6Bit)0 ,
/* local and remote port */
            TM_SELECTOR_WILD_PORT,      TM_SELECTOR_WILD_PORT,
/* protocol */
            TM_SELECTOR_WILD_PROTOCOL,
        },
/* selector #1, From 1.1.1.0/24 to 2.2.2.1, any port, any protocol */
        {
            "1.1.1.0",          (int)"255.255.255.0",
            "2.2.2.1",          0,
            TM_SELECTOR_LOCIP_SUBNET + TM_SELECTOR_REMTIP_HOST,
            TM_SELECTOR_WILD_PORT,      TM_SELECTOR_WILD_PORT,
            TM_SELECTOR_WILD_PROTOCOL,
        }
/* for the mask addr, you may want to use prefix length, especially when you
are using IPv6 address. In this case, you can write the selector in this way:
        {
            "1.1.1.0",          24,
            "2.2.2.1",          0,
            TM_SELECTOR_LOCIP_SUBNET + TM_SELECTOR_USE_PREFIX_LENGTH +
            TM_SELECTOR_REMTIP_HOST,
            TM_SELECTOR_WILD_PORT,      TM_SELECTOR_WILD_PORT,
            TM_SELECTOR_WILD_PROTOCOL,
        }
*/
    };
};
```

```

    ttPolicyContentInString plcyContent[] =
    {
/* policy content #0 */
    {
/* tunnel local IP, don't care if not for tunnel mode */
        (char*)0,
/* tunnel remote IP, don't care if not for tunnel mode */
        (char*)0,
/* policy rules */
        TM_PFLAG_BYPASS,
/* authentication algorithm */
        0,
/* encryption algorithm */
        0,
/* lifetime in seconds, if 0, we use the default value */
        0,
/* lifetime in kBytes, if 0, we use the default value */
        0
    },

/* policy content #1 */
    {
        "1.1.1.1",
        "2.2.2.1",
        TM_PFLAG_ESP + TM_PFLAG_TUNNEL,
        0,
        SADB_EALG_DESCBC,
        0,
        0
    },

/* policy content #2 */
    {
        (char*)0,
        (char*)0,
        TM_PFLAG_AH + TM_PFLAG_TRANSPORT,
        SADB_AALG_SHA1HMAC,
        0,
        0,
        0
    }
};

/* This is a very important table. This specifies which policy
 * rules are applied to received packets (TM_IPSEC_INBOUND), sent

```

```
* packets (TM_IPSEC_OUTBOUND),AND IN WHAT ORDER.
*
* Which policy rule(s) to use
* We screen the traffic using selectors which could be fine-
* grained or coarse grained. We specify different policy rules
* in the name of policy_content. The traffic_selector and
* policy_content is a one-to-one or one-to-many matching pair. For
* example <selector#0, content#0, INBOUND+OUTBOUND> is a matching
* pair which means all traffic screened by selector#0 is subject to
* the policy rule content#0, and this matching pair applies to both
* INBOUND and OUTBOUND traffic. Another matching pair example
* <selector#1, content#1+content#2, INBOUND+OUTBOUND> means all
* traffic, both INBOUND and OUTBOUND, screened by selector#1 is
* subject to the policy rule content#1 first AND then policy
* content#2.
*/
    ttIpsecPolicyPair plcyPair[] =
    {
/* <selector #0, content#0, INBOUND+OUTBOUND> */
        {0, 0, TM_IPSEC_BOTH_DIRECTION},
/* <selector #1, content#1 + content #2, INBOUND+OUTBOUND> */
        {1, 1, TM_IPSEC_BOTH_DIRECTION},
        {1, 2, TM_IPSEC_BOTH_DIRECTION}
/* The following matching pair is for explanation only */
/* <selector #2, content#3, OUTBOUND> */
/* {2, 3, TM_IPSEC_OUTBOUND} */
    };

/* Selector#1 has two continuous entries in the matching pair list,
* which means they are bundled policy contents and they must
* work together to protect the traffic. The inner policy content
* {selector#1, content#1, TM_IPSEC_BOTH_DIRECTION} will be applied
* first, followed by the outer policy content {selector#1,
* content#2, TM_IPSEC_BOTH_DIRECTION}.
*
* In what order to apply matching criteria
* The above matching pair list is navigated in REVERSE order.
* Matching pair at the bottom of the list is applied first (NOTE:
* Bundled matching entries are treated as a whole one). If the
* traffic failes to match the bottom selector, it tries the one
* above it, and so on until it finds a match. (If no match found,
* packet will be dropped.) For the above matching pair list, all
* outbound traffic will try to match <selector#2, content#3,
* OUTBOUND> first, if fails(the selector#2 doesn't apply to the
```

```
* traffic), try to match the bundled matching pair <selector#1,  
* content#1+content#2, INBOUND+OUTBOUND>, if still fails, try to  
* match <selector#0, contnet#0, INBOUND+OUTBOUND> and bypass the  
* traffic. If the matching of bundled pair successes, the traffic  
* will be protected by content#1 first and then content#2.  
*/  
  
    errorCode = tfPolicyRestore(plcyPair,  
        plcySelector,  
        plcyContent,  
/* Total entries of the matching pair list, don't consider the  
* explanation entry  
*/  
        3);  
}
```

4.5.2 tfPolicyAdd

```
ttPolicyEntryPtr    tfPolicyAdd
(
ttIpssecSelectorPtr selectorPtr,
ttPolicyContentPtr  contentPtr,
ttUser8Bit          direction,
int TM_FAR *        errorPtr
);
```

Description

Although **tfPolicyRestore** is recommended to initialize the SPD, the user may call **tfPolicyAdd** to add more policy entries into the SPD during the running time. Since **tfPolicyAdd** is designed for internal use, we do not use the string format IP address for the parameters, instead, all the IP addresses involved in the structures should be `sockaddr_storage`, which can store either IPv4 or IPv6 address.

Parameters

<i>selectorPtr</i>	Notice that <i>selectorPtr</i> is of type <code>ttIpssecSelectorPtr</code> , not of type <code>ttIpssecSelectorInString</code> pointer. tfPolicyAdd is designed for internal use, the IP addresses is not in string format, instead, in <code>sockaddr_storage</code> format.
<i>contentPtr</i>	Notice that <i>contentPtr</i> is of type <code>ttPolicyContentPtr</code> , not of type <code>ttPolicyContentInString</code> pointer. The difference is that the user can not use “1.1.1.1” or “iipsec.Treck.com” kind of source or destination IP address, rather, they must be a 32Bit network order data.
<i>direction</i>	Specifies which direction this policy applies to. Valid value could be <code>TM_IPSEC_INBOUND</code> , <code>TM_IPSEC_OUTBOUND</code> , or <code>TM_IPSEC_BOTH_DIRECTION</code> .
<i>errorPtr</i>	Error code pointer

Returns

Value	Meaning
policy pointer	Success
NULL pointer	Error code is stored in <code>errorPtr</code> .

Example

This example adds an OUTBOUND policy at gateway 1.1.1.1:

From 1.1.1.0/24 to 2.2.2.1, any port, any protocol, ESP tunnel from 1.1.1.1 to 2.2.2.1. Using DES-CBC for encryption, and NULL authentication. Share the SA for all hosts inside subnet 1.1.1.0/24

```
{
    ttIpssecSelector  selector;
    ttPolicyContent   content;
    ttPolicyEntryPtr  plcyPtr;
    int               errorCode;

    /* initialize the memory */
    tm_bzero(&selector, sizeof(ttIpssecSelector));
    tm_bzero(&content, sizeof(ttPolicyContent));
```

```

/* set the selector */
    selector.selLocIp1.ss_family = PF_INET;
    selector.selLocIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selLocIp2.ss_family = PF_INET;
    selector.selLocIp2.ss_len = sizeof(struct sockaddr_in);
    selector.selRemtIp1.ss_family = PF_INET;
    selector.selRemtIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selLocIp1.addr.ipv4.sin_addr.s_addr = inet_addr("1.1.1.0");
    selector.selLocIp2.addr.ipv4.sin_addr.s_addr = inet_addr("255.255.255.0");
    selector.selRemtIp1.addr.ipv4.sin_addr.s_addr = inet_addr("2.2.2.1");
    selector.selRemtPort = TM_SELECTOR_WILD_PORT;
    selector.selLocPort = TM_SELECTOR_WILD_PORT;
    selector.selProtocol = TM_SELECTOR_WILD_PROTOCOL;
    selector.selIpFlags = TM_SELECTOR_LOCIIP_SUBNET + TM_SELECTOR_REMTIP_HOST;
/* local ip is a subnet address, remote ip is a host address */

/* set the policy content */
    ccontent.pctLocIpAddr.ss_family = PF_INET;
    content.pctLocIpAddr.ss_len = sizeof(struct sockaddr_in);
    content.pctRemtIpAddr.ss_family = PF_INET;
    content.pctRemtIpAddr.ss_len = sizeof(struct sockaddr_in);
    content.pctLocIpAddr.addr.ipv4.sin_addr.s_addr = inet_addr("1.1.1.1");
/* tunnel source*/
    content.pctRemtIpAddr.addr.ipv4.sin_addr.s_addr = inet_addr("2.2.2.1");
/*tunnel dest*/
    content.pctRuleFlags = TM_PFLAG_ESP + TM_PFLAG_TUNNEL;
    content.pctEncryptAlg = SADB_EALG_DESCBC;
    content.pctAuthAlg = 0; /* NULL authentication algorithm*/

/* call tfPolicyAdd to add OUTBOUND policy. Actually, if this policy
 * is symmetric, i.e. applies to both INBOUND and OUTBOUND, user may
 * use TM_IPSEC_BOTH_DIRECTION instead of TM_IPSEC_OUTBOUND. The
 * user needs to change nothing, because we are using Local and
 * Remote, rather than Source and Destination
 */
    plcyPtr = tfPolicyAdd(&selector, &content, TM_IPSEC_OUTBOUND, &errorCode);

/* ... */

}

```

NOTE: *tfPolicyAdd can add one policy (with only one policy content) each time. To add a bundled policy content, you must use tfPolicyAddBundle.*

The IPsec kernel makes a copy of the selectorPtr and contentPtr, the user may free them after the call.

4.5.3 tfPolicyAddBundle

```
int                tfPolicyAddBundle
(
  ttPolicyEntryPtr  plcyPtr,
  ttPolicyContentPtr  outerContentPtr
);
```

Description

tfPolicyAddBundle adds an outer policy content to the specified policy. The resulting policy will have at least two policy contents. These policy contents must work together to protect the traffic. They are bundled policy contents.

Parameters

Value	Meaning
<i>plcyPtr</i>	The pointer to the policy interested.
<i>outerContentPtr</i>	The policy content we are going to add. This content will be linked to the content linked list, and it will become the outmost policy content.

Returns

Value	Meaning
TM_ENOERROR	policy successfully added
Other value	the call is unseccessful

NOTE: *The policy must have already at least one policy content in order to call tfPolicyAddBundle. Otherwise, call tfPolicyAdd instead. The IPsec kernel will save a copy of this content - user may free outerContentPtr after the call. The outerContentPtr will be of the same direction attribute as the inner policy content. The outerContentPtr's content flag must not conflict with the inner content flags, for example, you can't add outer policy content to a BYPASS or DISCARD policy.*

4.5.4 tfPolicyDelete

```
int          tfPolicyDelete
(
ttPolicyEntryPtr  plcyPtr,
tt8Bit           direction
);
```

Description

tfPolicyDelete deletes a policy of the specified direction. If the policy was of two directions, i.e. applying to both inbound and outbound, only the specified direction is removed and the policy entry remains in the SPD. Otherwise, if the policy was applicable to only the specified direction, the policy entry will be freed.

Parameters

Value	Meaning
<i>plcyPtr</i>	The pointer to the policy interested.
<i>direction</i>	can be any of TM_IPSEC_INBOUND, TM_IPSEC_OUTBOUND or TM_IPSEC_BOTHDIRECTION

Returns

Value	Meaning
TM_ENOERROR	policy successfully added
Other value	unsuccessful

NOTE: All SA spawned by this policy and in this direction will be deleted!

4.5.5 `tfIpsecPolicyQueryBySelector`

```
int                tfIpsecPolicyQueryBySelector
(
ttUser32Bit       options,
ttIpsecSelectorPtr pktSelectorPtr,
ttPolicyEntryPtrPtr plcyPtrPtr,
ttUser8Bit        direction
);
```

Description

This function locates the policy used to protect packets that have selector value `pktSelectorPtr`. Per RFC requirement, the first matched policy will be returned.

Parameters

Value	Meaning
<i>options</i>	This variable is not used currently.
<i>pktSelectorPtr</i>	The selector value of interested packet.
<i>plcyPtrPtr</i>	For output use. Carries the address of policy just found.
<i>direction</i>	Specifies inbound or outbound policy, can be either <code>TM_IPSEC_INBOUND</code> or <code>TM_IPSEC_OUTBOUND</code> , (CANNOT be <code>TM_IPSEC_BOTHDIRIRECTION</code> here)

Returns

Value	Meaning
<code>TM_ENOERROR</code>	Policy located successfully. Policy pointer stored in <code>plcyPtrPtr</code> .
Other value	Error code, and <code>plcyPtrPtr</code> carries a NULL pointer

Example

This example tries to locate the OUTBOUND policy that is used to protect TCP packet from 1.1.1.100 port 3500 to 2.2.2.1 port 21.

```
{
    ttIpsecSelector selector;
    ttPolicyEntryPtr plcyPtr;
    int                errorCode;

    /* initialize the memory */
    tm_bzero(&selector, sizeof(ttIpsecSelector));

    /* set the selector */
    selector.selLocIp1.ss_family = PF_INET;
    selector.selLocIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selRemtIp1.ss_family = PF_INET;
    selector.selRemtIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selLocIp1.addr.ipv4.sin_addr.s_addr = inet_addr("1.1.1.100");
    selector.selRemtIp1.addr.ipv4.sin_addr.s_addr = inet_addr("2.2.2.1");
    selector.selRemtPort = htons(21);
    selector.selLocPort = htons(3500);
    selector.selProtocol = TM_IP_TCP;
    selector.selIpFlags = TM_SELECTOR_LOCIIP_HOST + TM_SELECTOR_REMTIP_HOST;
```

```
/* call tfIpsecPolicyQueryBySelector */
    errorCode = tfIpsecPolicyQueryBySelector(0, &selector, &plcyPtr,
    TM_IPSEC_OUTBOUND);

/* ... */

}
```

***NOTE:** This function is called by TCP/IP kernel to determine the policy used. It is listed in public document for debug usage*

4.5.6 `tfIpsecPolicyQueryByIndex`

```
int          tfIpsecPolicyQueryByIndex
(
int          index,
ttPolicyEntryPtrPtr  plcyPtrPtr
);
```

Description

All policies have an index number, beginning from zero. This procedure gets the policy given its index number.

Parameters

Value	Meaning
<i>index</i>	The interested policy index
<i>plcyPtrPtr</i>	For output use. Carries the address of policy just found.

Returns

Value	Meaning
TM_ENOERROR	Policy located successfully. Policy pointer stored in <i>plcyPtrPtr</i> .
Other value	Error happens, <i>plcyPtrPtr</i> carries NULL pointer, no policy found.

4.5.7 tfPolicyClear

```
void                tfPolicyClear (void);
```

Description

This function clears the SPD and deletes all related SA's as well. The user must add a valid policy after this call in order for IPsec to work properly.

Parameters

None

Returns

None

NOTE: All SA's will be cleared consequently. If TM_USE_IPSEC is defined, and no policy table exists at all, no traffic will be allowed no matter inbound or outbound.

4.6 SA Database Management API

If `TM_USE_IKE` is defined, the SAD management, such as adding and deleting an SA, will be done by IKE automatically. If manual keying is used, the following API will be used to access the SAD hash table.

4.6.1 tfSadbRecordGenerate

```
int          tfSadbRecordGenerate
(
ttSadbRecordPtrPtr    saPtrPtr,
ttChildSaInfoPtr     saInfoPtr,
ttUser8BitPtr        keyMatPtr,
ttUser16Bit          keyMatLen,
ttUser8Bit           direction
);
```

Description

This function call has been replaced by `tfSadbRecordManualAdd`. User is recommended to use `tfSadbRecordManualAdd` instead. See section 4.5.9.

This procedure is used to generate one SA. The SA information will be included in a `ttChildSaInfo` pointer `saInfoPtr` and the key material is given by `keyMatPtr`. The SA will be inserted into the SAD hash table. Comparing to `tfSadbRecordManualAdd`, user can specify key length for each algorithm and variable lifetime parameters. This is useful if user want to add SA using non-default key length or specify the lifetime.

Parameters

Value	Meaning
<code>saPtrPtr</code>	For output use. Carries the address of SA just generated.
<code>saInfoPtr</code>	Input, has most of the information of what kind of SA we are going to generate, see section Structure References for the fields of structure <code>ttChildSaInfo</code>
<code>keyMatPtr</code>	Pointer to the keying material
<code>keyMatLen</code>	Length of the keying material
<code>direction</code>	Pdirection of the SA we are going to generate, can be either <code>TM_IPSEC_INBOUND</code> or <code>TM_IPSEC_OUTBOUND</code> . <code>TM_IPSEC_BOTH_DIRECTION</code> is not supported, because the keying material is used for one direction only.

Returns

Value	Meaning
<code>TM_ENOERROR</code>	an SA is successfully constructed. <code>saPtrPtr</code> contains the new SA pointer's address.
Other value	Error happens, <code>saPtrPtr</code> will contain NULL pointer

Example

Let's use the `tfPolicyAdd` example again:

From 1.1.1.0/24 to 2.2.2.1, any port, any protocol, ESP tunnel from 1.1.1.1 to 2.2.2.1. Using DES-CBC for encryption, and NULL authentication. **Share** the SA for all hosts inside subnet 1.1.1.0/24

Since sharing SA is granted, we need one single outbound SA for all traffic from 1.1.1.0/24 to 2.2.2.1, suppose we already has added the policy, as we did in the example `tfPolicyAdd`, now we begin to add an outbound SA.

```
{
    ttChildSaInfo    sainfo;
    tt8Bit           keyMat[] =
        "abcdefghijklmnopqrstabcdefghijklmnopqrstuvw";
}
```

```

ttSadbRecordPtr sadbPtr;

/* initialize memory */
tm_bzero(&sainfo, sizeof(ttChildSaInfo));

/* set Selector value, the selector value for the SA */
sainfo.chdPacketSelector.selLocIp1.ss_family = PF_INET;
sainfo.chdPacketSelector.selLocIp1.ss_len = sizeof(struct sockaddr_in);
sainfo.chdPacketSelector.selLocIp2.ss_family = PF_INET;
sainfo.chdPacketSelector.selLocIp2.ss_len = sizeof(struct sockaddr_in);
sainfo.chdPacketSelector.selRemtIp1.ss_family = PF_INET;
sainfo.chdPacketSelector.selRemtIp1.ss_len = sizeof(struct sockaddr_in);
sainfo.chdPacketSelector.selRemtIp1.addr.ipv4.sin_addr.s_addr =
    inet_addr("2.2.2.1");
sainfo.chdPacketSelector.selLocIp1.addr.ipv4.sin_addr.s_addr =
    inet_addr("1.1.1.0");
sainfo.chdPacketSelector.selLocIp2.addr.ipv4.sin_addr.s_addr =
    inet_addr("255.255.255.0");
sainfo.chdPacketSelector.selRemtPort = TM_SELECTOR_WILD_PORT;
sainfo.chdPacketSelector.selLocPort = TM_SELECTOR_WILD_PORT;
sainfo.chdPacketSelector.selProtocol =
    TM_SELECTOR_WILD_PROTOCOL;
sainfo.chdPacketSelector.selIpFlags =
    TM_SELECTOR_LOCIP_SUBNET + TM_SELECTOR_REMTIP_HOST;

/* set other fields of sainfo: the information of child SA, i.e. the
 * IPsec SA
 */

sainfo.chdAuthKeyBytes = 0; /* for NULL auth algorithm*/
sainfo.chdEncryptKeyBytes = 8; /* 8 bytes key for DES-CBC*/
sainfo.chdLifetimeKbytes = 0 ; /* not supported yet */
sainfo.chdLifetimeMinutes = 60; /* time value lifetime */
sainfo.chdMySpi = 1000; /* my inbound SPI, peer -> I use this */
sainfo.chdPeerSpi = 1500; /* my outbound SPI, I -> peer use*/
sainfo.chdPlcyPtr = plcyPtr; /* we already have this */
sainfo.chdPlcyContentPtr = plcyPtr->plcyContentPtr;

/* add SA for given SA information, keying material is 44 bytes
 * long, we need only the first 8 bytes for this case. That doesn't
 * matter
 */
errorCode = tfSadbRecordGenerate
    (&sadbPtr, &sainfo, keyMat, 44, TM_IPSEC_OUTBOUND);
...
}

```

NOTE: Do not use `TM_IPSEC_BOTH_DIRECTION` for the direction variable. Different direction will be using different keying material (resulting from different SPI, see RFC 2409 for details).

4.6.2 `tfSadbRecordDelete`

```
int                tfSadbRecordDelete
(
  ttUser32Bit      options,
  ttSadbRecordPtr sadbPtr
);
```

Description

This procedure is called to delete an SA.

Parameters

<i>options</i>	zero if the SAD is not locked by the calling procedure, or <code>TM_IPSEC_DATABASE_LOCKED</code> if SAD is already locked by the calling procedure. For manual usage, this is always zero.
<i>sadbPtr</i>	Pointer to the SA we want to delete.

Returns

Value	Meaning
<code>TM_ENOERROR</code>	SA is successfully deleted
Other value	unsuccessful operation, SA is not deleted

4.6.3 `tfSadbRecordDeleteByPolicy`

```
int          tfSadbRecordDeleteByPolicy
(
ttUser32Bit  options,
ttPolicyEntryPtr plcyPtr,
ttUser8Bit   direction
);
```

Description

Call this function to delete all SA's that spawned by the specified policy and match the specified direction.

Parameters

Value	Meaning
<i>options</i>	zero if the SAD is not locked by the calling procedure, or <code>TM_IPSEC_DATABASE_LOCKED</code> if SAD is already locked by the calling procedure. For manual usage, this is always zero.
<i>plcyPtr</i>	Pointer to the policy of which we want to delete all the spawned SA's.
<i>direction</i>	This can be <code>TM_IPEC_INBOUND</code> , <code>TM_IPSEC_OUTBOUND</code> or <code>TM_IPSEC_BOTH_DIRECTION</code> . SA's spawned by this policy, and matching this direction, will be all deleted. If only one direction is specified, the other direction SA's will be retained.

Returns

Value	Meaning
<code>TM_ENOERROR</code>	SA's were successfully deleted
Other value	Unsuccessful operation

4.6.4 tfSadbRecordDeleteByDestination

```
int  tfSadbRecordDeleteByDestination
(
ttUser32Bit                               options,
struct sockaddr_storage TM_FAR *          destination
);
```

Description

This procedure is called to delete all SA's destined to the destination IP address.

Parameters

Value	Meaning
<i>options</i>	zero if the SAD is not locked by the calling procedure, or TM_IPSEC_DATABASE_LOCKED if SAD is already locked by the calling procedure. For manual usage, this is always zero.
<i>destination</i>	The destination IP address, specified in struct sockaddr_storage format. By this means, we will keep this same API for IPv6.

Returns

Value	Meaning
TM_ENOERROR	SA's were successfully deleted
Other value	Unsuccessful operation

Example

This example shows how to delete all SA's destined to "2.2.2.1"

```
{
    struct sockaddr_storage  ipstorage;
    int                      errorCode;

    tm_bzero(&ipstorage, sizeof(struct sockaddr_storage));
    ipstorage.ss_family = PF_INET;
    ipstorage.ss_len = sizeof(struct sockaddr_in);
    ipstorage.addr.ipv4.sin_addr.s_addr = inet_addr("2.2.2.1");

    errorCode = tfSadbRecordDeleteByDestination(0, &ipstorage);
    ...
}
```

4.6.5 tfSadbRecordClear

```
void          tfSadbRecordClear
(
ttUser32Bit   options
);
```

Description

This function is called to clear the SAD table. All SAs, including inbound and outbound, will be deleted.

Parameters

Value	Meaning
<i>options</i>	Zero if the SAD is not locked by the calling procedure, or TM_IPSEC_DATABASE_LOCKED if SAD is already locked by the calling procedure. For manual usage, this is always zero.

Returns

None

4.6.6 tfSadbRecordGet

```
int                tfSadbRecordGet
(
  ttUser32Bit      options,
  ttSaIdentityPtr  saidPtr,
  ttSadbRecordPtrPtr  sadbPtrPtr
);
```

Description

This function is called to find an SA using the triple <destination address, protocol, SPI>. It is usually called by the INBOUND path where we know the triple value. This function is designed for kernel usage, however, user may call it directly for debugging purpose.

Parameters

Value	Meaning
<i>options</i>	Zero if the SAD is not locked by the calling procedure, or TM_IPSEC_DATABASE_LOCKED if SAD is already locked by the calling procedure. For manual usage, this is always zero.
<i>saidPtr</i>	Contains the triple value
<i>sadbPtrPtr</i>	For output use. Carries the address of the located SA pointer.

Returns

Value	Meaning
TM_ENOERROR	SA is successfully located. saPtrPtr contains the SA pointer's address
TM_EWRONGSPI	SPI value is invalid
-1	SA is not found

Example

This example locates SA which has the triple value of <destination = "1.1.1.1", ESP, SPI=1000>.

```
{
    ttSaIdentity      said;
    ttSadbRecordPtr  saPtr;
    int               errorCode;

    said.siDstSockaddr.ss_family = PF_INET;
    said.siDstSockaddr.ss_len = sizeof(struct sockaddr_in);
    said.siDstSockaddr.addr.ipv4.sin_addr.s_addr = inet_addr("1.1.1.1");
    said.siSpi = 1000;
    said.siProto = TM_IP_ESP;

    errorCode = tfSadbRecordGet(0, &said, &saPtr);
...
}
```

4.6.7 tfSadbRecordFind

```

int          tfSadbRecordFind
(
ttUser32Bit  options,
ttIpsecSelectorPtr pktSelectorPtr,
ttSadbRecordPtrPtr sadbPtrPtr,
ttPolicyEntryPtr plcyPtr,
ttPolicyContentPtr plcyContentPtr,
ttUser8Bit  direction
);

```

Description

This function is called to find a SA. It is usually called by the outbound path (it is also available for inbound) to find the SA to protect packets that match selector *pktSelectorPt* , given the policy and the policy content. The SA is returned in *sadbPtrPtr*.

Parameters

Value	Meaning
<i>options</i>	Zero if the SAD is not locked by the calling procedure, or TM_IPSEC_DATABASE_LOCKED if SAD is already locked by the calling procedure. For manual usage, this is always zero.
<i>pktSelectorPtr</i>	The selector value of the given packet
<i>sadbPtrPtr</i>	For output use. The located SA pointer address will be stored in <i>sadbPtrPtr</i> .
<i>plcyPtr</i>	The policy pointer.
<i>plcyContentPtr</i>	The policy content pointer.
<i>direction</i>	This is normally TM_IPSEC_OUTBOUND, anyway, for manual keying, you may want to set it to be TM_IPSEC_INBOUND, to locate an inbound SA and delete that SA or modify it.

Returns

Value	Meaning
TM_ENOERROR	An SA is successfully located. <i>saPtrPtr</i> contains the SA pointer
error code	<i>saPtrPtr</i> contains an NULL pointer

Example

This example attempts to locate the OUTBOUND **Authentication Header** SA for a TCP packet from 1.1.1.100 port 3500 to 2.2.2.1 port 21 (suppose our policy has bundled policy contents, the inner most policy content is for ESP, and the outermost policy content is for AH).

This example attempts to locate the OUTBOUND **Authentication Header** SA for a TCP packet from 1.1.1.100 port 3500 to 2.2.2.1 port 21 (suppose our policy has bundled policy contents, the inner most policy content is for ESP, and the outermost policy content is for AH).

```

{
    ttIpsecSelector  selector;
    ttPolicyEntryPtr plcyPtr;
    ttSadbRecordPtr  saPtr;
    int              errorCode;
}

```

```
/* initialize the memory */
    tm_bzero(&selector, sizeof(ttIpsecSelector));

/* set the selector */
    selector.selLocIp1.ss_family = PF_INET;
    selector.selLocIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selRemtIp1.ss_family = PF_INET;
    selector.selRemtIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selLocIp1.addr.ipv4.sin_addr.s_addr = inet_addr("1.1.1.100");
    selector.selRemtIp1.addr.ipv4.sin_addr.s_addr = inet_addr("2.2.2.1");
    selector.selRemtPort = htons(21);
    selector.selLocPort = htons(3500);
    selector.selProtocol = TM_IP_TCP;
    selector.selIpFlags = TM_SELECTOR_LOCIIP_HOST + TM_SELECTOR_REMTIP_HOST;

/* call tfIpsecPolicyQueryBySelector to locate the policy, as our assumption
 * suggests, this policy should have bundled policy contents - inner ESP content
 * and outer AH content
 */
    errorCode = tfIpsecPolicyQueryBySelector(0, &selector, &plcyPtr,
TM_IPSEC_OUTBOUND);
    if(errorCode != TM_ENOERROR)
    {
        return *;
    }
/* locate the AH SA */
    errorCode = tfSadbRecordFind(0,
        &selector,
        &saPtr,
        plcyPtr,
/* note that the inner content is ESP content, we are looking for SA for the
 * outer, i.e. AH content
 */
        plcyPtr->plcyContentPtr->pctOuterContentPtr,
        TM_IPSEC_OUTBOUND);
    if(errorCode != TM_ENOERROR)
    {
        return *;
    }
...
}
```

4.6.8 `tfSadbRecordSetOptions`

```
void tfSadbRecordSetOptions
(
ttSadbRecordPtr          sadbPtr,
ttUser16Bit              optionType,
ttUser8Bit                optionValue
);
```

Description

This function is called to set some option value for a Security Association.

Parameters

Value	Meaning
<i>sadbPtr</i>	Pointer to the SA
<i>optionType</i>	Option type to set. Valid option types are: TM_IPSEC_SADB_NOREKEYING: indicates that we don't want to rekey this SA. When SA expires, just delete it. Any non zero optionValue will enable this option, a zero optionValue will disable this option. TM_IPSEC_SADB_NOEXPIRING: Indicates that we don't want to rekey nor delete this SA. Whenever SA expires, we just renew the timer. Any non zero optionValue will enable this option, a zero optionValue will disable this option.
<i>optionValue</i>	Typically 1 or 0, see the description of optionType.

Returns

Value	Meaning
none	

4.6.9 tfSadbRecordManualAdd

```
int tfSadbRecordManualAdd(ttSadbRecordPtrPtr      sadbPtrPtr,
                        ttUser8Bit                direction,
                        ttUser32Bit                spi,
                        ttIpsecSelectorInStringPtr selStrPtr,
                        int                         plcyContentIndex,
                        const char TM_FAR *        manualKeyPtr,
                        ttUser32Bit                keyLength)
```

Description

This procedure is used to generate one SA manually. User is responsible to provide all the information in order to generate this SA, such as the direction, SPI number, selector, the corresponding policy content, and key materials. The SA will be inserted into the SAD hash table. You may use `tfSadbRecordGenerate` to manually add IPsec SA too.

Parameters

Value	Meaning
<i>saPtrPtr</i>	For output use. Carries the address of SA just generated.
<i>direction</i>	The direction of the to-be-generated SA, can be either <code>TM_IPSEC_INBOUND</code> or <code>TM_IPSEC_OUTBOUND</code>
<i>spi</i>	The SPI value of the interested SA, must be greater than 255.
<i>selStrPtr</i>	Pointer to the selector information
<i>plcyContentIndex</i>	The value of policy content index. Each policy content in the SPD table has a unique policy content index. We use this index to find the corresponding policy
<i>manualKeyPtr</i>	Pointer to the key material, this key material should include everything needed to generate this SA. For example, if both encryption and hashing needed for this SA to be generated, the key material should include encryption key and hashing key. Encryption key comes first and hashing key is appended right after the encryption key. Note that you can only use default key length for each algorithm, if you want key length other than the default one, you have to call <code>tfSadbRecordGenerate</code> . For default key length, see section 5.1.2 & 5.1.3
<i>keyLength</i>	Key material length. This is the total length of both encryption key and hashing key. You are allowed to provide a key material longer than necessary.

Returns

Value	Meaning
<code>TM_ENOERROR</code>	an SA is successfully constructed. <i>saPtrPtr</i> contains the new SA pointer's address.
Other value	Error happens, <i>saPtrPtr</i> will contain NULL pointer

Example

Suppose our policy reads: From 1.1.1.0/24 to 2.2.2.1, any port, any protocol, ESP tunnel from 1.1.1.1 to 2.2.2.1. Using DES-CBC for encryption, and MD5-HMAC hashing. Different protocol uses different SA. We want to manually add SAs according to this policy.

Since different protocol is going to use different SA. Here, we just add SA to support UDP traffic. In order to protect TCP, ICMP etc, user is responsible to add the corresponding inbound and outbound SAs.

```
{
    tt8Bit          keyMatInbound[ ]=
        "abcdefghijklmnopqrstabcdefghijklmnopqrstuvw";
    tt8Bit          keyMatOutbound[ ]=
```

```

    "1234efghIJKLmnopGRSTabcdEFGHijklMNOPqrst4321";
    ttSadbRecordPtr sadbPtr;
    int          errorCode;
    int          policyContentIndex;
    ttIpsecSelectorInString saSelector =
/* SA selector: From 1.1.1.0/24 to 2.2.2.1, any port, UDP protocol, please notice that saSelector
is maybe different with the policySelector. They could be the same, but in
many cases, saSelector is more specific than policySelector. Such as this
example, saSelector MUST specify a unique protocol-UDP, but the
policySelector just specifies a TM_SELECTOR_WILD_PROTOCOL, see examples in
tfPolicyRestore. All protocol uses the same policy, but each protocol MUST
has its own IPsec SA, according to the policy content */
    {
        "1.1.1.0",          (int)"255.255.255.0",
        "2.2.2.1",          0,
        TM_SELECTOR_LOCIIP_SUBNET + TM_SELECTOR_REMTIP_HOST,
        TM_SELECTOR_WILD_PORT,    TM_SELECTOR_WILD_PORT,
        IPPROTO_UDP /* you CAN'T use wild value here to generate IPsec SA */
    }
/* add SA according to the policy content #1. In order to know the right policy
content index, please refer to the example in section 4.4.1 tfPolicyRestore*/
    policyContentIndex = 1;
/* add inbound SA, using SPI = 1000*/
    errorCode = tfSadbRecordManualAdd(&sadbPtr,
                                     TM_IPSEC_INBOUND,
                                     1000,
                                     &saSelector,
                                     policyContentIndex,
                                     keyMatInbound,
                                     44); /*DES-CBC + MD5-HMAC requires 8+ 16 =
24 bytes key material, you have to provide at least 24 bytes. 44 is OK. */
    if(errorCode)          goto COMMON_RETURN;
/* add outbound SA, using SPI = 1100*/
    errorCode = tfSadbRecordManualAdd(&sadbPtr,
                                     TM_IPSEC_OUTBOUND,
                                     1100,
                                     &saSelector,
                                     policyContentIndex,
                                     keyMatOutbound,
                                     44);
    if(errorCode)          goto COMMON_RETURN;
    ...
}

```

NOTE: Do not use `TM_IPSEC_BOTH_DIRECTION` for the direction variable. Different direction will be using different keying material (resulting from different SPI, see RFC 2409 for details).

4.7 Log Function API

4.7.1 tfUseIpsecLogging

```
int                tfUseIpsecLogging
(
  ttUser16Bit      numLogMsgs
);
```

Description

This function and **tfIpsecLogWalk** are two public API for packet logging using Ipsec, especially for packet filter/logging cases such as pass, pass-log, reject, reject-log, ipsec policy and ipsec policy-log. Pass and Reject constrains can be implemented by setting **TM_PFLAG_BYPASS** and **TM_PFLAG_DISCARD** policy flag, respectively. Packet Logging should be enabled by setting **TM_PFLAG_LOG** flag in the corresponding policy.

Each log packet will be stored in a circular buffer and includes its selector, policy flag and first 40 bytes starting from upper layer header (TCP, UDP, ICMP) as ASCII format (available output by printf).

tfUseIpsecLogging is to allocate the log message buffer and must be called after **tfUseIpsec** function. **tfIpsecLogWalk** is for going through the log buffer and running a callback function for each message.

Parameters

Value	Meaning
<i>numLogMsgs</i>	The maximum log message number in the circular buffer. The newest message will replace the oldest message if there is not empty space in the buffer.

Returns

Value	Meaning
TM_ENOERROR	Successful
TM_EIPSECNOTINITIALIZED	TM_USE_IPSEC is not defined, or tvIpsecPtr is not correctly allocated
TM_ENOBUFS	No more buffers available

4.7.2 tflpsecLogWalk

```
int          tfIpsecLogWalk
(
ttLogWalkFuncPtr  funcPtr,
int              msgSeqNo,
ttUserGenericUnion genParam
);
```

Description

Walk the messages in the ipsec circular log buffer in the sequence in which they were logged, optionally starting with a specified message sequence number. If the specified message sequence number is not found, then all messages are walked.

Parameters

Value	Meaning
<i>funcPtr</i>	Pointer to user-specified function called for each log message. This function returns 0 to continue the walk, otherwise returns non-zero to terminate the walk. The function prototype for the user-specified function is: <pre>int myLogWalk(int msgSeqNo, const char TM_FAR *pszLogMsg, ttUserGenericUnion genParam);</pre> where msgSeqNo is the 15-bit sequence number of message pointed by pszLogMsg , and pszLogMsg is a pointer to the null-terminated log message. The log is walked in the forward direction, therefore message sequence numbers will be in ascending order.
<i>msgSeqNo</i>	An optional 15-bit message sequence number specifying where to start the walk. Specify -1 if there is no specific sequence number, and instead you want to walk all messages.
<i>GenParam</i>	A generic parameter that is passed on to the funcPtr when it's called

Returns

Value	Meaning
TM_ENOERROR	Successful
TM_EPERM	The log buffer was not available.

4.8 Use Hardware Accelerator

IPsec calculations, including encryption/decryption, authentication, diffie-hellman exchanges, public key calculation (DSA or RSA), are all highly CPU intensive tasks. Hardware accelerator may be used to accelerate the process.

4.8.1 tfCryptoEngineRegister

```
int          tfCryptoEngineRegister
(unsigned int engineId,
ttUserVoidPtr initParamPtr,
ttCryptoEnginePtrPtr newEnginePtrPtr,
ttCryptoEngineInitFuncPtr engineInitFuncPtr,
ttCryptoGetRandomWordFuncPtr randomFuncPtr,
ttCryptoSessionOpenFuncPtr sessionOpenFuncPtr,
ttCryptoSessionFuncPtr sessionProcessFuncPtr,
ttCryptoSessionFuncPtr sessionCloseFuncPtr);
```

Description

This function registers a new crypto engine. If `TM_IPSEC_USE_SOFTWARE_CRYPTOTOENGINE` is defined, the TRECK IPsec initialization will automatically register a crypto engine whose `engineId` is `TM_CRYPTOTOENGINE_SOFTWARE`. Currently, we also support `engineId` `TM_CRYPTOTOENGINE_HIFN7951`, which denotes hardware crypto engine HIFN 7951.

Parameters

Value	Meaning
<i>engineId</i>	The name (ID) of crypto engine. Currently we support <code>TM_CRYPTOTOENGINE_SOFTWARE</code> which denotes TRECK's software crypto library. Another one is <code>TM_CRYPTOTOENGINE_HIFN7951</code> which denotes HIFN7951 hardware accelerator.
<i>initParamPtr</i>	Initialization parameter for the to-be-registered crypto engine. This argument may be NULL, if no parameter is needed to initialize the crypto engine.
<i>newEnginePtrPtr</i>	Pointer to the new engine pointer. Upon successfully return of this call, <code>newEnginePtrPtr</code> should store the new engine pointer.
<i>engineInitFuncPtr</i>	The function pointer to crypto engine initialization function. This function will take only one argument, <code>initParamPtr</code> . It could be NULL if no initialization function required.
<i>randomFuncPtr</i>	Random function of this crypto engine. If this function is NULL, we are going to use a default software implementation (<code>tfCryptoDefaultGetRandomWord</code>) to get random number.
<i>sessionOpenFuncPtr</i>	Function pointer to open a crypto session. A session, in TRECK's implementation, means the environment in which any crypto action is using the same algorithm(s) and same key material. An unexpired IPsec SA is an active session, when you create a new IPsec SA, you open a new session.
<i>sessionProcessFuncPtr</i>	Function pointer to process any crypto request to this crypto engine.
<i>sessionCloseFuncPtr</i>	Function pointer to close a crypto session. For example, when you close an IPsec SA or rekey an IPsec SA, you need to close the old crypto session.

Returns

Value	Meaning
<code>TM_ENOERROR</code>	New crypto engine is successfully registered. (<code>newEnginePtrPtr</code> stores the pointer to this new crypto engine.)
Other value	New crypto engine is not successfully registered.

Example

The following example registers a new crypto engine, TM_CRYPTO_ENGINE_HIFN7951. The correspond function is provided in trhf7951.c

```
{

    ttCryptoEnginePtrPtr    newEnginePtrPtr;
    int                     errorCode;
    ttHifn7951InitArgument  hifnInitParam;

    hifnInitParam.startAddressP = 0xa0000000;
    hifnInitParam.targetAddressP = 0x80000000;

    errorCode = tfCryptoEngineRegister(TM_CRYPTO_ENGINE_HIFN7951,
                                         (ttUserVoidPtr)&hifnInitParam,
                                         &hifnEnginePtr,
                                         tfHifn79xxInitialize,
                                         tfHifn7951GetRandomWord, /* hardware
random number generator */
                                         tfHifn7951SessionOpen,
                                         tfHifn7951SessionProcess,
                                         tfHifn7951SessionClose);
}
```

4.8.2 tfCryptoEngineDeRegister

```
int          tfCryptoEngineDeRegister  
(unsigned int engineId)
```

Description

This function deregisters a crypto engine registered before.

Parameters

Value	Meaning
<i>engineId</i>	The name (ID) of crypto engine. Currently we support TM_CRYPTTO_ENGINE_SOFTWARE which denotes TRECK's software crypto library. Another one is TM_CRYPTTO_ENGINE_HIFN7951 which denotes HIFN7951 hardware accelerator.

Returns

Value	Meaning
TM_ENOERROR	The crypto engine ID is successfully deregistered.
Other value	New crypto engine is not successfully registered.

4.8.3 tfCryptoEngineAddAlgorithm

```
int tfCryptoEngineAddAlgorithm
( ttCryptoEnginePtr      cenginePtr,
  unsigned int           algorithmId)
```

Description

This function adds supported algorithm to the crypto engine. Some crypto engine may support more algorithms than others. For example, the HIFN7951 crypto engine only supports MD5, SHA1, DES, 3DES, it does not support AES algorithm

Parameters

Value	Meaning
<i>cenginePtr</i>	Pointer to the crypto engine
<i>algorithmId</i>	Algorithms can be added to this crypto engine. Valid values are: SADB_AALG_NULL /* NULL hashing algorithm*/ SADB_AALG_MD5HMAC /* MD5 hashing algorithm*/ SADB_AALG_SHA1HMAC /* SHA1 hashing algorithm*/ SADB_AALG_RIPEMDHMAC /* RIPEMD hashing algorithm*/ SADB_AALG_SHA256HMAC /* sha256 hashing algorithm*/ SADB_EALG_NULL /* NULL encryption algorithm*/ SADB_EALG_DESIV64CBC /* DESIV64 encryption algorithm*/ SADB_EALG_DES_CBC /*DES algorithm*/ SADB_EALG_3DESCBC /*3DES algorithm*/ SADB_EALG_RC4CBC /* RC4 algorithm*/ SADB_EALG_RC5CBC /*RC5 algorithm*/ SADB_EALG_IDEACBC /* IDEA algorithm*/ SADB_EALG_CAST128CBC /*CAST128 algorithm*/ SADB_EALG_BLOWFISHCBC /* blowfish algorithm*/ SADB_EALG_3IDEACBC /*3IDEA algorithm*/ SADB_EALG_AESCBC /*aes algorithm*/ SADB_EALG_TWOFISHCBC /* two fish algorithm*/ SADB_PUBKEY_RSA /*RSA algorithm*/ SADB_PUBKEY_DSA /* DSA algorithm*/ SADB_PUBKEY_DIFFIEHELLEMAN1 /*DH group1*/ SADB_PUBKEY_DIFFIEHELLEMAN2 /*DH group 2*/ SADB_PUBKEY_DIFFIEHELLEMAN5 /* DH group 5*/ SADB_PUBKEY_X509 /*x509 certificate*/

Returns

Value	Meaning
TM_ENOERROR	The algorithm is successfully added
Other value	New crypto engine is not successfully registered.

Example

The following example registers a new crypto engine, TM_CRYPTO_ENGINE_HIFN7951. And then add MD5, SHA1, DES, 3DES, DiffieHellman group 1 and group 2 support.

```
{
    ttCryptoEnginePtrPtr      newEnginePtrPtr;
    int                       errorCode;
    ttHifn7951InitArgument    hifnInitParam;

    hifnInitParam.startAddressP = 0xa0000000;
    hifnInitParam.targetAddressP = 0x80000000;

    errorCode = tfCryptoEngineRegister(TM_CRYPTO_ENGINE_HIFN7951,
                                       (ttUserVoidPtr)&hifnInitParam,
                                       &hifnEnginePtr,
                                       tfHifn79xxInitialize,
                                       tfHifn7951GetRandomWord, /* hardware
random number generator */
                                       tfHifn7951SessionOpen,
                                       tfHifn7951SessionProcess,
                                       tfHifn7951SessionClose);

    errorCode = tfCryptoEngineAddAlgorithm(hifnEnginePtr, SADB_AALG_MD5HMAC);
    assert(errorCode == TM_ENOERROR)
    errorCode = tfCryptoEngineAddAlgorithm(hifnEnginePtr,
SADB_AALG_SHA1HMAC);
    assert(errorCode == TM_ENOERROR)
    errorCode = tfCryptoEngineAddAlgorithm(hifnEnginePtr, SADB_EALG_DESCBC);
    assert(errorCode == TM_ENOERROR)
    errorCode = tfCryptoEngineAddAlgorithm(hifnEnginePtr, SADB_EALG_3DESCBC);
    assert(errorCode == TM_ENOERROR)
    errorCode = tfCryptoEngineAddAlgorithm(hifnEnginePtr,
SADB_PUBKEY_DIFFIEHELLEMAN1);
    assert(errorCode == TM_ENOERROR)
    errorCode = tfCryptoEngineAddAlgorithm(hifnEnginePtr,
SADB_PUBKEY_DIFFIEHELLEMAN2);
    assert(errorCode == TM_ENOERROR)
}
```

5 Structure Reference

5.1 Cryptology Structures

5.1.1 ttGenericKey

```
typedef struct tsGenericKey
{
ttUser8BitPtr      keyDataPtr;
int                keyRounds;
ttUser32Bit        keyBits;
#if (defined(TM_IPSEC_USE_SW_CRYPTENGINE) || define(TM_USE_PPP_MSCHAP))
union{
    void TM_FAR * keySchedulePointer;
    void TM_FAR * keyContextPointer;
}schd_context_ptr;
union{
    ttUser16Bit keyScheduleLength;
    ttUser16Bit keyContextLength;
}schd_context_len;
#define keySchedulePtr schd_context_ptr.keySchedulePointer
#define keyContextPtr schd_context_ptr.keyContextPointer
#define keyScheduleLen schd_context_len.keyScheduleLength
#define keyContextLen schd_context_len.keyContextLength
#endif /*TM_IPSEC_USE_SW_CRYPTENGINE|| TM_USE_PPP_MSCHAP */
}
ttGenericKey;
typedef ttGenericKey * ttGenericKeyPtr ;
```

Structure Description

ttGenericKey holds both the raw key data and the schedule data in encryption algorithm or context data in authentication algorithm. If `TM_IPSEC_USE_SW_CRYPTENGINE` is not defined (i.e. use hardware accelerator), we don't need to maintain the schedule (or context)

Structure Members

Member	Description
<i>keyDataPtr</i>	Raw key data
<i>keySchedulePtr</i>	Schedule data pointer for encryption algorithm
<i>keyContextPtr</i>	Context data pointer for authentication algorithm. KeySchedulePtr and keyContextPtr are unioned together in union schd_context_ptr
<i>keyRounds</i>	Key rounds
<i>keyBits</i>	Raw key length measured in bits
<i>keyScheduleLen</i>	Schedule data length for encryption algorithm
<i>keyContextLen</i>	Context data length for authentication algorithm. KeyScheduleLen and keyContextLen are unioned together in union schd_context_len.

5.1.2 ttAhAlgorithm

```
typedef struct ttAhAlgorithm
{
ttUser16Bit      aalgName;
ttUser16Bit      aalgKeyMin;
ttUser16Bit      aalgKeyMax;
ttUser16Bit      aalgKeyDefault;
ttUser16Bit      aalgDigestOutBits;
ttUser16Bit      aalgDigestTruncateBits;
void             (TM_CODE_FAR *aalgInitFuncPtr) (ttUserVoidPtr  contextPtr);
void             (TM_CODE_FAR *aalgUpdateFuncPtr) (ttUserVoidPtr  contextPtr,
                                                    ttUser8BitPtr   moreDataPtr,
                                                    ttUser32Bit    size,
                                                    ttUser32Bit    offset);
void             (TM_CODE_FAR *aalgFinalFuncPtr) (ttUser8BitPtr  outDigest,
                                                    ttUserVoidPtr  contextPtr);
int              (TM_CODE_FAR *aalgContextLenFuncPtr) (void);
}
ttAhAlgorithm;
typedef ttAhAlgorithm * ttAhAlgorithmPtr;
```

Structure Description

Any authentication algorithm must follow the format of ttAhAlgorithm. The user may use this format to add more authentication algorithms

Structure Members

Member	Description
<i>aalgName</i>	Algorithm name, like SADB_AALG_MD5HMAC
<i>aalgKeyMin</i>	The minimum key size, in bits
<i>aalgKeyMax</i>	The maximum key size, in bits
<i>aalgKeyDefault</i>	The default key size, in bits
<i>aalgDigestOutBits</i>	The output digest size, in bits, for SHA1_HMAC, it is 160 bits, and for MD5_HMAC, it is 128 bits
<i>aalgDigestTruncateBits</i>	The truncated output digest size, in bits, for Authentication Header, it is always 96 bits.
<i>aalgInitFuncPtr</i>	The initialization function pointer. It takes one parameter, the context pointer. (should be cast to void pointer)
<i>aalgUpdateFuncPtr</i>	The update function pointer. It takes four parameters, the context pointer, data, datalength, and offset. (Offset is used for 16Bit or 32Bit DSP, when the pointer increases every 16 Bit or 32Bit, offset is 0, 1 or 0,1,2,3 respectively.
<i>aalgFinalFuncPtr</i>	The finalization function pointer. It takes two parameters, the output digest pointer and the context pointer.
<i>aalgContextLenFuncPtr</i>	Pointer to function which calculates the size of the corresponding context, like ttMd5Cxt or ttSha1Cxt, the length is returned

Example

This example attempts to add another authentication algorithm foobar_HMAC_96. foobar_HMAC_96 accepts key size 240 bits only. It originally outputs digest size 240 bits. We have the context structure for foobar_HMAC_96 algorithm ttFoobarContext and we wish to use 4 to indicate this authentication algorithm. Now we make foobar_HMAC_96 available in the AH algorithm list. The list is in trahcore.c. Before doing that, user should make

tfFoobarInit, tfFoobarUpdate, tfFoobarFinal, and tfFoobarContextLen ready.

```
static ttAhAlgorithm ahAlgorithms[] = {
    { SADB_AALG_NONE, 0, 0,0, 0, 0,
      0, 0, 0,
      0},
    { SADB_AALG_MD5HMAC, 128, 128, 128,128, 96,
      tfMd5Init, tfMd5Update, tfMd5Final,
      tfMd5ContextLen},
    { SADB_AALG_SHA1HMAC, 160, 160, 160, 160, 96,
      tfShalInit, tfShalUpdate, tfShalFinal,
      tfShalContextLen},
    { SADB_AALG_foobarHMAC, 240, 240, 240, 240, 96,
      tfFoobarInit, tfFoobarUpdate, tfFoobarFinal,
      tfFoobarContextLen} /* user add the red lines */
};
```

Now the user may use algorithm SADB_AALG_FOobarHMAC (4) the same way as SADB_AALG_MD5HMAC or SADB_AALG_SHA1HMAC.

The following table lists the key length requirement for Treck hashing algorithms and encryption algorithms.

Keylength (in bits)	min	max	default
Hashing Algorithms			
MD5:	128	128	128
SHA1:	160	160	160
Ripemd:	160	160	160
Encryption/Decryption Algorithms:			
DES:	64	64	64
3DES:	192	192	192
Blowfish:	40	448	128
Cast128:	40	128	128
AES:	128	256	128
TWOFISH:	128	256	128
(AES and Twofish keylength can be 128, 192 or 256 only)			

5.1.3 ttEspAlgorithm

```

typedef struct  ttEspAlgorithm
{
int             ealgName;
ttUser8Bit     ealgBlockSize; /* block size, in byte */
ttUser8Bit     ealgPaddingFlag;
ttUser16Bit    ealgKeyMin; /* in bits */
ttUser16Bit    ealgKeyMax; /* in bits */
ttUser16Bit    ealgKeyDefault; /* use for IKE */
int            (TM_CODE_FAR *ealgScheduleLenFuncPtr) (void);
int            (TM_CODE_FAR *ealgScheduleFuncPtr)
                (struct ttEspAlgorithm TM_FAR * ealgPtr,
                 ttGenericKeyPtr keyPtr);
int            (TM_CODE_FAR *ealgBlockDecryptFuncPtr)
                (struct ttEspAlgorithm TM_FAR * ealgPtr,
                 ttGenericKeyPtr keyPtr,
                 tt8BitPtr srcPtr,
                 tt8BitPtr dstPtr);
int            (TM_CODE_FAR *ealgBlockEncryptFuncPtr)
                (struct ttEspAlgorithm TM_FAR * ealgPtr,
                 ttGenericKeyPtr keyPtr,
                 ttUser8BitPtr srcPtr,
                 ttUser8BitPtr dstPtr);
}
ttEspAlgorithm;
typedef ttEspAlgorithm * ttEspAlgorithmPtr;

```

Structure Description

Any encryption algorithm must follow the format of ttEspAlgorithm. User may use this format to add more encryption algorithms. See examples of ttAhAlgorithm. For encryption algorithm, the user needs to modify espAlgorithms in file tresp.c.

Structure Members

Member	Description
<i>ealgName</i>	Algorithm name, like SADB_EALG_DESCBC
<i>ealgBlockSize</i>	Block size (in bytes) of this algorithm, generally, it is either 8 or 16 bytes.
<i>ealgPaddingFlag</i>	Indicates how to pad the ESP, use minimum length or random length. Currently, set this flag to one, and we don't want to support random size padding at this time.
<i>ealgKeyMin</i>	The minimum key size, in bits. See previous page for the whole list
<i>ealgKeyMax</i>	The maximum key size, in bits. See previous page for the whole list
<i>ealgKeyDefault</i>	The default key size, in bits. See previous page for the whole list
<i>ealgScheduleLenFuncPtr</i>	Pointer to function which computes the schedule length of this algorithm
<i>ealgScheduleFuncPtr</i>	Pointer to function which computes the schedule. It takes two parameters, the algorithm pointer and a generic key pointer. This function will allocate memory for the schedule using the size computed from ealgScheduleLenFuncPtr, and calculates the schedule and saved into the generic key.
<i>ealgBlockDecryptFuncPtr</i>	Pointer to function which decrypts a single block data from src to dst using the keyPtr.

ealgBlockEncryptFuncPtr

Pointer to function which encrypts a single block data from src to dst using the keyPtr.

5.2 Policy Management Structures

5.2.1 ttlpsecSelectorInString

```
typedef struct tsIpsecSelectorInString
{
    char    TM_FAR    * selstrLocIp1;
    union
    {
        int            selstr_loc_prefix_length;
        char TM_FAR    * selstr_loc_ip2;
    } selstrLocIp2Union;
    char TM_FAR    * selstrRemtIp1;
    union
    {
        int            selstr_remt_prefix_length;
        char TM_FAR    * selstr_remt_ip2;
    } selstrRemtIp2Union;

    ttUser16Bit      selstrIpFlags;
/* local port and remote port */
    ttUser16Bit      selstrLocPort;
    ttUser16Bit      selstrRemtPort;
    ttUser16Bit      selstrProtocol;

#define selstrLocPrefixLength selstrLocIp2Union.selstr_loc_prefix_length
#define selstrLocIp2          selstrLocIp2Union.selstr_loc_ip2
#define selstrRemtPrefixLength selstrRemtIp2Union.selstr_remt_prefix_length
#define selstrRemtIp2          selstrRemtIp2Union.selstr_remt_ip2
} ttlpsecSelectorInString;
```

Structure Description

When the user wishes to call **tfPolicyRestore** to bulk-load the policy entries, the user must fill out the selectors. Selector defines the matching criteria including the IP addresses, port number, and protocols. For selectors, we are using local and remote side, rather than source and destination because the selector applies to both inbound traffic and outbound traffic.

Structure Members

Member	Description
<i>selstrLocIp1</i>	With selstrLocIp2Union, selstrLocIp1 specifies the IP information for the local side. If local IP selector is a single IP, we need selstrLocIp1 only (we don't care what is selstrLocIp2Union); if local IP selector is a subnet, then selstrLocIp1 indicates the subnet address, and selstrLocIp2Union indicates the mask or the prefix length; if local IP selector is an IP range, then selstrLocIp1 is minimum end and selstrLocIp2Union is the maximum end.
<i>selstrLocIp2Union</i>	See selstrLocIp1. If this union is assigned an integer value, it means the prefix length (user need specify this in selstrIpFlags), Otherwise, if it is assigned a char pointer, it points to an IP address string.
<i>selstrRemtIp1</i>	With selstrRemtIp2Union, selstrRemtIp1 specifies the IP information for the remote side.

<i>selstrRemtIp2Union</i>	See selstrRemtIp1 and selstrLocIp2Union
<i>selstrIpFlags</i>	Specifies the IP selector type of both the local side and the remote side. Valid values are: <pre> /* local ip selector is a host IP address */ TM_SELECTOR_LOCIIP_HOST 0x01 /*remote ip selector is a host IP address */ TM_SELECTOR_REMTIP_HOST 0x02 /*both ip selectors are host IP addresses */ TM_SELECTOR_BOTHIP_HOST 0x03 /* = 0x01 0x02 */ /*local ip selector is a subnet*/ TM_SELECTOR_LOCIIP_SUBNET 0x04 /*remote ip selector is a subnet*/ TM_SELECTOR_REMTIP_SUBNET 0x08 /*both ip selectors are subnets*/ TM_SELECTOR_BOTHIP_SUBNET 0x0c /* = 0x04 0x08 */ /*local ip selector is an ip range(min, max)*/ TM_SELECTOR_LOCIIP_RANGE 0x10 /*remote ip selector is an ip range(min, max)*/ TM_SELECTOR_REMTIP_RANGE 0x20 /*both ip selectors are ip ranges(min, max)*/ TM_SELECTOR_BOTHIP_RANGE 0x30 /* = 0x10 0x20 */ /*local ip selector is a domain name*/ TM_SELECTOR_LOCIIP_DNS_NAME 0x40 /*remote ip selector is a domain name, must be a host name, no * subnet, no ip ranges */ TM_SELECTOR_REMTIP_DNS_NAME 0x80 /* both ip selector are domain name, must be a host name, no * subnet, no ip ranges */ TM_SELECTOR_BOTHIP_DNS_NAME 0xc0 /* = 0x40 0x80 */ TM_SELECTOR_USE_PREFIX_LENGTH 0x100 /*Use prefix length to specify a subnet */ </pre>
<i>selstrLocPort</i>	Local side port number, in host order. Could be TM_SELECTOR_WILD_PORT (0) to indicate this field is a wild value
<i>selstrRemtPort</i>	Remote side port number, in host order. Could be TM_SELECTOR_WILD_PORT (0) to indicate this field is a wild value
<i>selstrProtocol</i>	The protocol number, Could be TM_SELECTOR_WILD_PROTOCOL (0) to indicate this field is a wild value

Examples

```

/* selector example 1, ANY to ANY */
ttIpsecSelectorInString sell =
{
    (char*)0,          0,
    (char*)0,          0,
    (tt16Bit)0 ,
    TM_SELECTOR_WILD_PORT,          TM_SELECTOR_WILD_PORT,
    TM_SELECTOR_WILD_PROTOCOL

```

```
};
```

```
/* selector example 2, local side 1.1.1.0/24 to remote side 2.2.2.1, any port, any protocol
*/
```

```
ttIpsecSelectorInString sel2 =
{
    "1.1.1.0",                (int)"255.255.255.0", /*cast to int to avoid
compiler warning message */
    "2.2.2.1",                0,
    TM_SELECTOR_LOCIIP_SUBNET + TM_SELECTOR_REMTIP_HOST,
    TM_SELECTOR_WILD_PORT,    TM_SELECTOR_WILD_PORT,
    TM_SELECTOR_WILD_PROTOCOL
};
```

This selector can also be written as:

```
ttIpsecSelectorInString sel2 =
{
    "1.1.1.0",                24,
    "2.2.2.1",                0,
    TM_SELECTOR_LOCIIP_SUBNET + TM_SELECTOR_USE_PREFIX_LENGTH +
TM_SELECTOR_REMTIP_HOST,
    TM_SELECTOR_WILD_PORT,    TM_SELECTOR_WILD_PORT,
    TM_SELECTOR_WILD_PROTOCOL
};
```

```
/* selector example 3, local side "foo1.bar1.com"any port to remote side "foo2.bar2.com",
* port 21, TCP protocol
*/
```

```
ttIpsecSelectorInString sel3 =
{
    "foo1.bar1.com",        0,
    "foo2.bar2.com",        0,
    TM_SELECTOR_BOTHIP_DNSNAME,
    TM_SELECTOR_WILD_PORT,    21,
    TM_IP_TCP
};
```

5.2.2 ttlpsecSelector

```
typedef struct tsIpsecSelector
{
    struct sockaddr_storage    selLocIp1;
    struct sockaddr_storage    selLocIp2;
    struct sockaddr_storage    selRemtIp1;
    struct sockaddr_storage    selRemtIp2;
    ttUser16Bit                selIpFlags;
    ttUser8Bit                 selProtocol;
    ttUser8Bit                 selPadding;
}
ttIpsecSelector;
```

Structure Description

ttIpsecSelectorInString is for the user's reference, so the IP information is the string format IPv4 or IPv6 address. ttlpsecSelector is designed for internal use, thus, all the IP addresses are in sockaddr_storage format, which can store either IPv4 or IPv6 socket address. And for internal use, we use an 8 bit value to indicate a protocol, resulting in an 8bit padding which we do not use.

Structure Members

Member	Description
<i>selLocIp1</i>	With selLocIp2, selLocIp1 specifies the local side IPv4 or IPv6 address, of type struct sockaddr_storage. If local side is an IP host address, we don't care the value in selLocIp2. If local IP selector is a subnet, then selLocIp1 indicates the subnet address and selLocIp2 indicates the mask. If local IP selector is an IP range, then selLocIp1 is minimum end and selLocIp2 is the maximum end.
<i>selLocIp2</i>	See selLocIp1
<i>selRemtIp1</i>	With selRemtIp2, selRemtIp1 specifies the IP information for the remote side.
<i>selRemtIp2</i>	See selRemtIp1
<i>selIpFlags</i>	See selstrIpFlags in ttlpsecSelectorInString. Anyway, it can't be any of the DNS flags, and prefix length, if any, will be translated to mask address.
<i>selLocPort</i>	Same as selLocPort in ttlpsecSelectorInString
<i>selRemtPort</i>	Same as selRemtPort in ttlpsecSelectorInString
<i>selProtocol</i>	Same as selProtocol in ttlpsecSelectorInString
<i>selPadding</i>	For alignment usage, not used.

5.2.3 ttPolicyContentInString

```
typedef struct tsPolicyContentInString
{
char TM_FAR *          pctstrLocIpAddr;
char TM_FAR *          pctstrRemtIpAddr;
ttUser32Bit           pctstrRuleFlags;
ttUser16Bit           pctstrAuthAlg;
ttUser16Bit           pctstrEncryptAlg;
ttUser32Bit           pctstrLifeSeconds;
ttUser32Bit           pctstrLifeKbytes;
}
ttPolicyContentInString;
```

Structure Description

This structure will be used when calling `tfPolicyRestore`. It specifies the policy contents.

Structure Members

Member	Description
<i>pctstrLocIpAddr</i>	Tunnel address at the local side. We use local and remote rather than source and destination because the policy content also applies to inbound traffic. Must be IPv4 or IPv6 address string, like "1.1.1.1" or "3FFF::1". If this policy content is for transport mode, this member variable is not used. The user may use (char*)0.
<i>pctstrRemtIpAddr</i>	Tunnel address at the remote side. Must be IPv4 or IPv6 address string. For transport mode policy content, this member is not used.
<i>pctstrRuleFlags</i>	Specifies the policy rules. This is a 32 bit value, and the bits are defined as the following way: (the lowest significant bit is bit 0) Bit 0: set means tunnel mode Bit 1: set means transport mode, you can't set both tunnel and transport Bit 2: set means AH Bit 3: set means ESP. You can't set both AH and ESP, you must use another policy content to achieve both AH and ESP. Bit 4: set means each distinct remote ip needs its own SA. Prohibit IP address sharing. See RFC 2401, 4.4.1 Bit 5: set means each distinct remote port needs its own SA. Prohibit port sharing. Bit 6: set means each distinct local ip needs its own SA. Prohibit local ip sharing. Bit 7: set means each distinct local port needs its own SA. Seldom used. Bit 8: set means each distinct protocol needs its own SA. Prohibit protocol sharing Bit 9: set means bypass ipsec policy check Bit 10: set means discard the matched packet Bit 11: set means the local IP uses a DNS name. Bit 12: set means that the remote IP uses a DNS name. Bit 13: set means log the matched traffic, see <code>tfUseIpsecLogging</code> Bit 14: This policy is used for the tunnelling interface between MN's Care of address and Home Agent

According to the above, the following position mask macros are defined:

TM_PFLAG_TUNNEL	0x01 /* bit 0 */
TM_PFLAG_TRANSPORT	0x02 /* bit 1 */
TM_PFLAG_AH	0x04 /* bit 2 */
TM_PFLAG_ESP	0x08 /* bit 3 */
TM_PFLAG_RIP_PACKET	0x10 /* bit 4 */
TM_PFLAG_RPT_PACKET	0x20 /* bit 5 */
TM_PFLAG_LIP_PACKET	0x40 /* bit 6 */
TM_PFLAG_LPT_PACKET	0x80 /* bit 7 */
TM_PFLAG_PROTO_PACKET	0x0100 /* bit 8 */
TM_PFLAG_BYPASS	0x0200 /* bit 9 */
TM_PFLAG_DISCARD	0x0400 /* 10 */
TM_PFLAG_LOCIP_DNS_NAME	0x0800 /* 11 */
TM_PFLAG_REMTIP_DNS_NAME	0x1000 /* 12 */
TM_PFLAG_LOG	0x2000 /* 13 */
TM_PFLAG_MIPV6_HA_TUNNEL	0x4000 /* 14 */

To specify AH tunnel, `pctRuleFlags = TM_PFLAG_AH + TM_PFLAG_TUNNEL`

pctstrAuthAlg

Specifies the authentication algorithm used. Valid value are:

SADB_AALG_NONE	0 /* NULL auth algorithm*/
SADB_AALG_MD5HMAC	2 /* md5_hmac_96*/
SADB_AALG_SHA1HMAC	3 /* sha1_hmac_96*/
SADB_AALG_RIPEMDHMAC	5 /* ripemd_hmac_96*/

pctstrEncryptAlg

Specifies the encryption algorithm used. Valid value are:

SADB_EALG_NONE	0 /* = NULL algorithm*/
SADB_EALG_DESCBC	2
SADB_EALG_3DESCBC	3
SADB_EALG_RC5CBC	4
SADB_EALG_CAST128CBC	6
SADB_EALG_BLOWFISHCBC	7
SADB_EALG_NULL	11 /* = NULL algorithm*/
SADB_EALG_RIJNDAELCBC	12 /* i.e. AES algorithm*/
SADB_EALG_TWOFISHCBC	13

SADB_EALG_NONE and SADB_EALG_NULL will be treated the same way.

IDEA algorithm is not supported simply because of the patent problem.

DES_IV64 is same as DES.

pctstrLifeSeconds

Specifies the lifetime in seconds of IPsec SAs which are generated according to this policy content. If the value is zero, we will use the default value 43200 seconds, i.e. 12 hours. The value should be no less than `TM_IPSECSA_TLIFETIME_MIN` (120 seconds) and no greater than `TM_IPSECSA_TLIFETIME_MAX` (43200 seconds).

pctstrLifeKbytes

Specifies the lifetime in kilo-bytes of IPsec SAs which are generated according to this policy content. If the value is zero, we will use the default value 32k. The value should be no less than `TM_IPSECSA_VLIFETIME_MIN` (10 kbytes) and no greater than `TM_IPSECSA_VLIFETIME_MAX` (32k-kbytes = 32M).

Examples

```

/* examples 1, BYPASS policy content */
ttPolicyContentInString pc1 =
{
    (char*)0,
    (char*)0,
    TM_PFLAG_BYPASS,
    0,
    0,
    0,
    0
};

/* Example 2, ESP tunnel, local side "3fff::1", remote "3fff::2",
 * using DES_CBC + MD5_HMAC authentication algorithm. SA uses policy
 * value (sharing SA as much as possible)
 */
ttPolicyContentInString pc2 =
{
    "3fff::1",
    "3fff::2",
    TM_PFLAG_ESP + TM_PFLAG_TUNNEL,
    SADB_AALG_MD5HMAC,
    SADB_EALG_DESCBC,
    0, /* use the default lifetime value in seconds*/
    0 /* use the default lifetime value in kbytes*/
};

/* Example 3, AH transport, using SHA1_HMAC authentication
 * algorithm. The protocol and remote IP address uses packet value
 * (means for each distinct protocol and each distinct remote IP
 * address, uses different SA), log the traffic. IPsec SA generated
 * according to this policy content will have lifetime 300 seconds
 * or 100kilo-bytes, whichever comes first.
 */
ttPolicyContentInString pc3 =
{
    (char *)0,
    (char *)0,
    TM_PFLAG_AH + TM_PFLAG_TRANSPORT + TM_PFLAG_PROTO_PACKET +
    TM_PFLAG_RIP_PACKET + TM_PFLAG_LOG,
    SADB_AALG_SHA1HMAC,
    0,
    300, /* use lifetime value 300 seconds */
    100 /* use lifetime value 100 kilo-bytes */
};

```

5.2.4 ttPolicyContent

```
typedef struct tsPolicyContent
{
struct tsPolicyContent TM_FAR * pctOuterContentPtr;
struct sockaddr_storage      pctLocIpAddr;
struct sockaddr_storage      pctRemtIpAddr;
ttUserPacketPtr             pctQueuePtr;
ttUserPacketPtr             pctLastQueuePtr;
ttUser32Bit                  pctRuleFlags;
ttUser32Bit                  pctLifeSeconds;
ttUser32Bit                  pctLifeKbytes;
ttUser32Bit                  pctContentIndex;
ttUser16Bit                  pctQueueBytes;
ttUser8Bit                   pctAuthAlg;
ttUser8Bit                   pctEncryptAlg;
}
ttPolicyContent;
```

Structure Description

ttPolicyContentInString is for user's reference, so the IP information is the string format IPv4 or IPv6 address. While ttPolicyContent is designed for internal use, thus all the IP addresses are in sockaddr_storage format. We have a pointer which points to the outer policy content if there is any. The two 8-bit padding is not used at all, just for alignment usage.

Structure Members

Member	Description
<i>pctOuterContentPtr</i>	Points to the outer policy content, if there is any. The inner policy content and all outer policy contents must work together to protect the traffic. They are bundled policy contents. If there is no outer policy content, this pointer is NULL.
<i>pctLocIpAddr</i>	Tunnel address at the local side. We use local and remote rather than source and destination, because the policy content will also applies to inbound traffic. Must be a sockaddr_storage structure to indicate either IPv4 or IPv6 address. If this policy content is for transport mode, this member variable is not used.
<i>pctRemtIpAddr</i>	Tunnel address at the remote side. See pctLocIpAddr above.
<i>pctQueuePtr</i>	Pointer to the first queued packet which is waiting for the establishment of security association according to this policy content, other queued packeted are link using pktChainNextPtr.
<i>pctLastQueuePtr</i>	Pointer to the last item of the queued packet list, i.e. the newest coming packet.
<i>pctRuleFlags</i>	Same as pctRuleFlags of ttPolicyContentInString. It may not be any of the DNS flags anyway.
<i>pctLifeSeconds</i>	Lifetime value in seconds of the IPsec SA for this policy content.
<i>pctLifeKbytes</i>	Lifetime value in kilo-bytes of the IPsec SA for this policy content.
<i>pctContentIndex</i>	The index number of this content.
<i>pctQueueBytes</i>	Indicates how much data is waiting for IKE to negotiate IPsec SA according to this policy content. The maximum number is TM_IKE_PACKET_MAX_QUEUE_BYTES
<i>pctAuthAlg</i>	Same as pctAuthAlg of ttPolicyContentInString. We use eight bit here. It is enough for up to 256 authentication algorithms.
<i>pctEncryptAlg</i>	Same as pctEncryptAlg of ttPolicyContentInString.

5.2.5 ttlpsecPolicyPair

```
typedef struct tsIpsecPolicyPair
{
    ttUser16Bit          selectorIndex;
    ttUser8Bit           plcycontentIndex;
    ttUser8Bit           direction;
}
ttIpsecPolicyPair;
```

Structure Description

ttPolicyPair will be used when the user calls **tfPolicyRestore**. ttPolicyPair specifies which rules are applied to received packets (TM_IPSEC_INBOUND), sent packets (TM_IPSEC_OUTBOUND), and *in what order*. It means any *direction* packet which matches the selector[*selectorIndex*] will be protected by policy_content[*plcycontentIndex*].

Structure Members

Member	Description
<i>selectorIndex</i>	Indicates the index of the selector lists. The selector list is initialized as ttIpsecSelectorInString type.
<i>plcyContentIndex</i>	Indicates the index of the policy content lists. The content list is initialized as ttPolicyContentInString type.
<i>direction</i>	Indicates the direction of this policy pair applies. Could be TM_IPSEC_INBOUND for inbound packet (receive path), or TM_IPSEC_OUTBOUND for outbound packet (send path), or TM_IPSEC_BOTH_DIRECTION for both receive path and send path.

Examples

```
/* Before this initialization, user should have initialized
 * selector[0~2], and policycontent[0~3]
 */
ttIpsecPolicyPair ppair[] =
{
    {0, 0, TM_IPSEC_BOTH_DIRECTION},
    {1, 1, TM_IPSEC_BOTH_DIRECTION},
    {1, 2, TM_IPSEC_BOTH_DIRECTION},
    {2, 3, TM_IPSEC_BOTH_DIRECTION}
};

/* This list is navigated in reverse order, rules at the bottom of
 * the list are matched and applied first. (bundled policy content
 * entries are treated as a whole one.)
 * plcyPair associates a specific matching criteria (selectors) with
 * a specific rule (the policy contents), and most importantly
 * specifies the order in which the match/rule combinations are
 * applied both to inbound/received and outbound/sent packets.
 * In this case, we are saying that for any (applies to both
 * direction) packet we want to apply the match/rule combination
 * selector[2]/
```

```
* policycontent[3] first. If the packet doesn't match the
* selector[2], *we should try to match selector[1], if they match,
* we need to apply
* the BUNDLED policy contents policycontent[1] and policycontent[2].
* (policycontent[1] is the innermost one and should apply first.) If
* the packet doesn't match selector[1], we need try selector[0], if
* they match, policycontent[0] applies.
*/
```

5.2.6 ttPolicyEntry

```
typedef struct tsPolicyEntry
{
    struct tsPolicyEntry  TM_FAR  * plcyNextPtr;
    struct tsSadbRecord   TM_FAR  * plcyInboundSadbPtr;
    struct tsSadbRecord   TM_FAR  * plcyOutboundSadbPtr;
    ttIpsecSelector       plcySelector;
    ttPolicyContentPtr    plcyContentPtr;
    ttUser16Bit           plcyIndex;
    ttUser16Bit           plcyHdrLen;
    ttUser16Bit           plcyTrailerLen;
    ttUser8Bit            plcyDirection;
    ttUser8Bit            plcyAction;
}
ttPolicyEntry;
typedef ttPolicyEntry * ttPolicyEntryPtr;
```

Structure Description

ttPolicyEntry is the structure for Treck IPsec policies. Generally, it is an internal structure, however, the user is able to locate a policy and modify it (see Policy Database Management APIs).

Structure Members

Member	Description
<i>plcyNextPtr</i>	All policies are linked together. This pointer points to the next policy entry.
<i>plcyInboundSadbPtr</i>	All inbound SA's spawned by this policy are linked together. This pointer points to the head of inbound SA list spawned by this policy.
<i>plcyOutboundSadbPtr</i>	All outbound SA's spawned by this policy are linked together. This pointer points to the head of outbound SA list spawned by this policy.
<i>plcySelector</i>	The matching criteria for this policy, i.e. the selector.
<i>plcyContentPtr</i>	All bundled policy contents are linked together. This pointer points to the innermost policy content.
<i>plcyIndex</i>	The index of this policy. Starting from zero. The higher the index is, the more preferred the policy is. (i.e., we try to match the policy with highest index first.)
<i>plcyHdrLen</i>	Indicates how many header bytes required to implement this policy content. For example, an AH transport policy content requires 24 octets as the head.
<i>plcyTrailerLen</i>	Indicates how many trailer bytes required to implement this policy content. For example, an ESP transport policy content with NULL authentication algorithm requires 8 octets as the trailer, considering the ESP tail and minimum padding.
<i>plcyDirection</i>	Indicates the policy is an TM_IPSEC_INBOUND policy or TM_IPSEC_OUTBOUND policy, or both direction TM_IPSEC_BOTH_DIRECTION.
<i>plcyAction</i>	Indicates what IPsec action we need to take. Valid value are: TM_IPSEC_POLICY_IPSEC 2 TM_IPSEC_POLICY_BYPASS 3 TM_IPSEC_POLICY_DISCARD 4 The above action number may or may not come with TM_IPSEC_POLICY_LOG_BASE 100 For example, if plcyAction = 104, then we must discard the packet and log the corresponding information.

5.3 SA Management Structures

5.3.1 ttSadbRecord

```
typedef struct tsSadbRecord
{
    struct tsSadbRecord  TM_FAR    * sadbNextSameHashPtr ;
    struct tsSadbRecord  TM_FAR    * sadbNextSamePolicyPtr;
    struct tsSadbRecord  TM_FAR    * sadbTwinSaPtr;
    struct tsPolicyEntry TM_FAR    * sadbPolicyPtr;
    struct tsPolicyContent TM_FAR  * sadbPlcyContentPtr;
    ttUser8BitPtr       sadbReplayWindowMapPtr ;
    struct tsIpsecSelector TM_FAR  * sadbSelectorPtr;
    ttUserVoidPtr       sadbSoftTimerPtr;
    ttUserVoidPtr       sadbHardTimerPtr;
    ttUserVoidPtr       sadbSessionPtr;
    ttUser32Bit         sadbRemainBytes;
    struct sockaddr_storage sadbSrcIpAddress;
    struct sockaddr_storage sadbDstIpAddress;
    struct tsGenericKey  sadbAuthKey;
    struct tsGenericKey  sadbEncryptKey;
    struct ttAhAlgorithm  TM_FAR    * sadbAuthAlgPtr;
    struct ttEspAlgorithm TM_FAR    * sadbEncryptAlgPtr;
    struct tsCryptoEngine TM_FAR    * sadbCryptoEnginePtr;
    sadb_sa               TM_FAR    * sadbSAPtr;
    ttUser32Bit          sadbSequenceNumber;
    ttUser32Bit          sadbReplayWindowBegin ;
    ttUser16Bit          sadbFlags;
    ttUser16Bit          sadbCheckOut;
    ttUser8Bit           sadbDirection;
    ttUser8Bit           sadbSaProtocol;
    ttUser8Bit           sadbIpsecMode;
    ttUser8Bit           sadbPadding;
}
ttSadbRecord ;
typedef ttSadbRecord      *ttSadbRecordPtr ;
```

Structure Description

ttSadbRecord is the structure for Treck IPsec SA's. Generally, it is an internal structure. However, the user is able to locate a SA and modify it (see SA Database Management APIs).

Structure Members

Member	Description
<i>sadbNextSameHashPtr</i>	All SA's are stored in a hash table. The pointer points to the next SA which has the same hash value
<i>sadbNextSamePolicyPtr</i>	All SA's spawned by the same policy (and in the same direction) are linked together. This pointer points to the next SA of this list.
<i>sadbTwinSaPtr</i>	If this SA is an inbound (or outbound)SA, sadbTwinSaPtr points to the corresponding outbound(or inbound) SA.
<i>sadbPolicyPtr</i>	Points to the policy which spawns this SA
<i>sadbPlcyContentPtr</i>	Points to the policy content which spawns this SA. (A policy may have several bundled contents, each content will result in a separate SA.
<i>sadbReplayWindowMapPtr</i>	Replay window size usage map. We use 8Bit array to store the map.

<i>sadbSelectorPtr</i>	Points to the selector used to match this SA.
<i>sadbSoftTimerPtr</i>	Soft lifetime pointer. It is time to rekeying , the time event will trigger a rekeying session
<i>sadbHardTimerPtr</i>	Hard lifetime pointer. Delete this SA when hard time reaches. Time event will trigger a delete action.
<i>sadbSessionPtr</i>	Session pointer, may be used for hardware accelertor.
<i>sadbRemainBytes</i>	Indicates how many bytes this SA can be used to protect. Used only if volume type lifetime is used.
<i>sadbSrcIpAddress</i>	Source IP address. SA can be either inbound SA or outbound SA, i.e. uni-directional, source and destination won't cause confusion (policy may apply to both directions, thus we use local and remote.)
<i>sadbDstIpAddress</i>	Destination IP address
<i>sadbAuthKey</i>	Authentication key
<i>sadbEncryptKey</i>	Encryption Key. (or decryption key)
<i>sadbAuthAlgPtr</i>	Authentication algorithm pointer
<i>sadbEncryptAlgPtr</i>	Encryption algorithm pointer
<i>sadbCryptoEnginePtr</i>	Pointer to the crypto engine, currently, may be software engine, or HIFN 7951
<i>sadbSAPtr</i>	pointer to a structure <code>sadb_sa</code> , which looks <pre> typedef struct sadb_sa { tt32Bit sadb_sa_spi ; /* SPI of this SA*/ tt8Bit sadb_sa_replay ; /*replay size*/ tt8Bit sadb_sa_state ; /* state of this SA, SADB_SASTATE_DYING means this SA reaches its soft lifetime. SADB_SASTATE_MATURE means in good condition to use */ tt8Bit sadb_sa_auth ; /* authentication algorithm, see pctAuthAlg for ttPolicyContentInString */ tt8Bit sadb_sa_encrypt ; /* encryption algorithm, see pctEncryptAlg for ttPolicyContentInString*/ } sadb_sa ; </pre>
<i>sadbSequenceNumber</i>	Sequence number of this SA at current time. When a 32Bit sequence number is used up, rekeying action should be taken
<i>sadbReplayWindowBegin</i>	Indicates the beginning sequence of the current replay window
<i>sadbFlags</i>	Flags of this SA. May be: TM_IPSEC_SADBFLAG_NOREKEYING: Means not to rekey this SA. When lifetime expires, delete this SA. TM_IPSEC_SADBFLAG_BEENUSED: Means that this SA has been used. If an SA is not used at all, we won't rekey it. TM_IPSEC_SADBFLAG_NOEXPIRING: Means that this SA won't expire. Once

	the current lifetime expires, it will automatically renew the timer..
<i>sadbCheckOut</i>	Indicates how many sockets checked out this SA. (For TCP sockets, once a SA is located, they will queue this SA for later use.)
<i>sadbDirection</i>	Indicates the direction of this SA, can be either TM_IPSEC_INBOUND or TM_IPSEC_OUTBOUND
<i>sadbSaProtocol</i>	Indicates the IPsec protocol. Can be either TM_IP_AH or TM_IP_ESP
<i>sadbIpsecMode</i>	Indicates the IPsec protocol operation mode, can be either SADB_MODE_TRANSPORT or SADB_MODE_TUNNEL
<i>sadbPadding</i>	For alignment padding. Not used.

5.3.2 ttSaIdentity

```
typedef struct tsSaIdentity
{
    struct sockaddr_storage siSrcSockAddr;
    struct sockaddr_storage siDstSockAddr;
    ttUser32Bit             siSpi;
    ttUser8Bit              siProto;
}
ttSaIdentity;
typedef ttSaIdentity * ttSaIdentityPtr;
```

Structure Description

ttSaIdentity is the structure to hold the triple value <Destination, IPsec protocol, SPI> which uniquely determines an SA. We also put siSrcSockAddr here, we may need to use the source address to find if we have IKE phase 1 SA with regarding to that address.

Structure Members

Member	Description
<i>siDstSockAddr</i>	Stores the IPv4 or IPv6 socket address of the source.
<i>siDstSockAddr</i>	Stores the IPv4 or IPv6 socket address of the destination.
<i>siSpi</i>	The SPI value of SA, in host order
<i>siProto</i>	Indicates the IPsec protocol, can be either TM_IP_AH or TM_IP_ESP

5.3.3 ttChildSaInfo

```
typedef struct tsChildSaInfo
{
    struct tsPolicyEntry TM_FAR *   chdPlcyPtr;
    struct tsPolicyContent TM_FAR * chdPlcyContentPtr;
    struct tsIpsecSelector          chdPacketSelector;
    ttSadbRecordPtr                chdRekeySaPtr;
    ttUser32Bit                    chdMySpi; /*in host order*/
    ttUser32Bit                    chdPeerSpi;
    ttUser32Bit                    chdLifetimeSeconds;
    ttUser32Bit                    chdLifetimeKbytes;
    ttUser16Bit                    chdAuthKeyBytes;
    ttUser16Bit                    chdEncryptKeyBytes;
}
ttChildSaInfo;
```

Structure Description

ttChildSaInfo has most information to construct an IPsec SA. If manually building SA, the user is responsible for building this structure first. If IKE is used to build SA, IKE is responsible for building this structure.

Structure Members

Member	Description
<i>chdPlcyPtr</i>	Pointer to the policy used
<i>chdPlcyContentPtr</i>	Pointer to the policy content used. A policy may have bundled policy contents. Each policy content results in a separate SA.
<i>chdPacketSelector</i>	Selector for the SA to be built. For IKE, IKE will generate this selector according to the information of policy rule flags and the triggering packet. For manual method, the user is responsible for building this selector. For example, if policy specifies to use packet value for protocol, then chdPacketSelector.selProtocol can NOT be TM_SELECTOR_WILD_PROTOCOL.
<i>chdRekeySaPtr</i>	Points to the IPsec SA which is going to be rekeyed.
<i>chdMySpi</i>	If the SA to be built is an inbound SA, use this SPI (which is picked by the local side itself). This value is in host order.
<i>chdPeerSpi</i>	If the SA to be built is an outbound SA, use this SPI (which is picked by the peer). In host order.
<i>chdAuthKeyBytes</i>	Indicates how many key bytes used for authentication algorithm
<i>chdEncryptKeyBytes</i>	Indicates how many key bytes used for encryption algorithm
<i>chdLifetimeSeconds</i>	Indicates the lifetime in seconds.
<i>chdLifetimeKbytes</i>	Indicates the lifetime in Kbytes.

5.4 Crypto Engine Structures

5.4.1 ttCryptoSessionOpenFuncPtr

```
typedef int (*ttCryptoSessionOpenFuncPtr)
(ttUserVoidPtr TM_FAR * sessionPtrPtr,
 ttUserVoidPtr param1Ptr,
 ttUserVoidPtr param2Ptr);
```

Structure Description

This function takes three parameters, and opens a crypto session.

Structure Members

Member	Description
<i>sessionPtrPtr</i>	When this function call returns, sessionPtrPtr stores the session pointer
<i>param1Ptr, param2Ptr</i>	Parameter to the function.

5.4.2 ttCryptoSessionFuncPtr

```
typedef int (*ttCryptoSessionFuncPtr)
(ttUserVoidPtr sessionPtr);
```

Structure Description

Session function.

Structure Members

Member	Description
<i>sessionPtr</i>	Pointer to the crypto session

5.4.3 ttCryptoEngineInitFuncPtr

```
typedef int (*ttCryptoEngineInitFuncPtr)
(ttUserVoidPtr initParamPtr);
```

Structure Description

Function pointer to the initialization function.

Structure Members

Member	Description
<i>initParamPtr</i>	The initialization parameter to the initialization function

5.4.4 ttCryptoGetRandomWordFuncPtr

```
typedef int (*ttCryptoGetRandomWordFuncPtr)
(ttUser32Bit TM_FAR * dataPtr,
 ttUser16Bit      wordSize);
```

Structure Description

This function get random word (four bytes).

Structure Members

Member	Description
<i>dataPtr</i>	Pointer to the place to hold the random number
<i>wordSize</i>	Random number size, in word (four-bytes) size.

5.4.5 tsCryptoEngine

```
typedef struct tsCryptoEngine
{
    ttUser32Bit                ceEngineId;
    ttUser32Bit                ceSupportFlags;
    ttCryptoEngineInitFuncPtr  ceInitFuncPtr;
    ttCryptoGetRandomWordFuncPtr ceRandomWordFuncPtr;
    ttCryptoSessionOpenFuncPtr ceSessionOpenFuncPtr;
    ttCryptoSessionFuncPtr     ceSessionProcessFuncPtr;
    ttCryptoSessionFuncPtr     ceSessionCloseFuncPtr;
}ttCryptoEngine;
```

Structure Description

This structure is the crypto engine structure.

Structure Members

Member	Description
<i>ceEngineId</i>	The crypto engine name (ID)
<i>ceSupportFlags</i>	Bit flags to indicate supporting algorithms
<i>ceInitFuncPtr</i>	The initialization function
<i>ceRandomWordFuncPtr</i>	The random number generator function.
<i>ceSessionOpenFuncPtr</i>	The function to open a crypto session
<i>ceSessionProcessFuncPtr</i>	The function to process crypto request for this crypto engine, which may belong to multiple sessions.
<i>ceSessionCloseFuncPtr</i>	The function to close a crypto session.

6 Examples

The IP addresses in the following examples are fictitious. User is responsible to substitute them with real addresses. We use IPv4 address as an example, IPv6 is similar. The examples assume user is already familiar with Treck TCP/IP stack for normal usage, and it is listed here emphasizing how to use IPsec and IKE only. (If the user does not wish to use IPsec or IKE, see the Treck User's Manual for other details.)

6.1 Manual keying

```
/*
```

6.1.1 Define the Policy

Since IPSEC will be used, for any traffic(even the bypass one), we need to have its policy.

Policies at 1.1.1.1, in the order of preference:

— Any protocol, any port traffic between 1.1.1.0/24 and 2.2.1.0/24, use ESP tunnel mode + AH transport mode to protect. ESP algorithm: 3DES-CBC+ MD5-HMAC-96 and, AH algorithm: SHA1-HMAC-96. Tunneled between 1.1.1.1 to 2.2.2.1, use packet value for upper layer protocols.

— Any to Any, BYPASS IPsec.

```
*/
```

```
/*
```

6.1.2 Set trsystem.h

Define TM_USE_IPSEC in trsystem.h

```
*/
```

```
#include <stdio.h>
```

```
#include <trsecapi.h>
```

```
#define TM_BUF_ARRAY_SIZE      4000
```

```
char    bufferArray[TM_BUF_ARRAY_SIZE];
```

```
char    bufferReceive[TM_BUF_ARRAY_SIZE];
```

```
static void testUdp(ttUserIpAddress clientIpAddr,  
                  ttUserIpAddress peerIpAddr,  
                  int iterations);
```

```
static int  addPolicyAndSa(void);
```

```
static int  testIpsecDifferentPolicy(ttUserIpAddress clientIpAddr,  
                                    ttUserIpAddress peerIpAddr,  
                                    int iterations);
```

```
int main ( )
```

```
{  
    ttUserLinkLayer linkLayerHandle;  
    ttUserInterface interfaceHandle;  
    ttUserIpAddress clientIPAddr;  
    ttUserIpAddress peerIPAddr;  
    ttUserIpAddress mask;  
    int              configFlags;  
    int              scatteredBufferCount;
```

```

int          errorCode;

int          iterations;
/* Allow scatter send */
configFlags = TM_DEV_SCATTER_SEND_ENB;
iterations = 100;
clientIPAddr = inet_addr("1.1.1.1");
peerIPAddr = inet_addr("2.2.2.1");
mask = inet_addr("255.255.255.0");

if (configFlags & TM_DEV_SCATTER_SEND_ENB)
{
    scatteredBufferCount = 1501;
}
else
{
    scatteredBufferCount = 1;
}

```

```

/*
6.1.3 Start Treck
*/

```

```

errorCode = tfStartTreck();
if (errorCode != TM_ENOERROR)
{
    printf( "tfStartTreck failed '%s'\n", tfStrError(errorCode));
    return errorCode;
}

```

```

/*
6.1.4 Add driver interface below Ethernet link layer
*/

```

```

linkLayerHandle = tfUseEthernet();
interfaceHandle = tfUseLinuxDriver("TEST", linkLayerHandle, &errorCode);

if (errorCode != TM_ENOERROR)
{
    printf( "Adding interface driver failed '%s'",
            tfStrError(errorCode) );
    return errorCode;
}

```

```

/*
6.1.5 Call tfUseIpsec

```

```

Initialize IPsec global variables
*/

```

```

errorCode = tfUseIpsec();
if (errorCode != TM_ENOERROR)
{
    printf( "tfUseIpsec failed '%s'\n");
    return errorCode;
}

```

```

/*

```

6.1.6 Add policy and SA

add SdbRecord should be done manually in current stage

```
*/
    errorCode = addPolicyAndSa ();
    if (errorCode != TM_ENOERROR)
    {
        printf( "addPolicyAndSa failed \n");
        return errorCode;
    }
}
```

```
/*
```

6.1.7 Open the added interface

```
*/
    errorCode = tfOpenInterface( interfaceHandle,
                                clientIPAddr,
                                mask,
                                configFlags,
                                scatteredBufferCount );
}
```

```
/* for IPv6, you may need to call tfNgOpenInterface*/
```

```
if (errorCode != TM_ENOERROR)
{
    printf("tfConfigInterface failed '%s'\n", tfStrError(errorCode));
    return errorCode;
}
```

```
/*
```

6.1.8 Test case

* Run different policy combinations

```
*/
    testIpsecDifferentPolicy(clientIPAddr, peerIPAddr, iterations);

    return errorCode;
}
```

```
/*
```

6.1.9 Procedure addPolicyAndSa

```
*/
static int addPolicyAndSa (void)
{
    const char    keyMat1[]="abcdefghijklmnopqrstabcdefghijklmnopqrstuvw";
    const char    keyMat2[]="13fhalsafdyijkafaslfdsfsafghijklmnopqrstuvw";
#ifdef TEST_USE_BUNDLE
    const char    keyMat3[]="idsaoaejnuoafjklaoipdsads";
    const char    keyMat4[]="afd09rewtjkl891trewxc9765";
#endif /* TEST_USE_BUNDLE */
    ttSadbRecordPtr    sadbPtr;
    int                errorCode = 0;
}
```

```
/*6.1.9.1 input selector, policy content, and policy matching pairs, call
tfPolicyRestore */
```

```

    ttIpsecSelectorInString plcySelector[] =
    {
/* selector #0, ANY to ANY */
    {
/* local ip and mask, or ipmin and ipmax if selector is a range */
        (char*)0,          0,
/* remote ip and mask, or ipmin and ipmax if selector is a range */
        (char*)0,          0,
/* ip flags */
        (ttl16Bit)0 ,
/* local and remote port */
        TM_SELECTOR_WILD_PORT,      TM_SELECTOR_WILD_PORT,
/* protocol */
        TM_SELECTOR_WILD_PROTOCOL,
    },
/* selector #1, From 1.1.1.0/24 to 2.2.2.1, any port, any protocol */
    {
        "1.1.1.0",          24,
        "2.2.2.1",          (char *)0,
        TM_SELECTOR_LOCIP_SUBNET + TM_SELECTOR_USE_PREFIX_LENGTH +
TM_SELECTOR_REMTIP_HOST,
        TM_SELECTOR_WILD_PORT,      TM_SELECTOR_WILD_PORT,
        TM_SELECTOR_WILD_PROTOCOL,
    },
/* selector #2, From 1.1.1.0/24 to 2.2.2.1, any port, UDP protocol */
    {
        "1.1.1.0",          24,
        "2.2.2.1",          (char *)0,
        TM_SELECTOR_LOCIP_SUBNET + TM_SELECTOR_USE_PREFIX_LENGTH +
TM_SELECTOR_REMTIP_HOST,
        TM_SELECTOR_WILD_PORT,      TM_SELECTOR_WILD_PORT,
        IPPROTO_UDP,
    }
    };

    ttPolicyContentInString plcyContent[] =
    {
/* policy content #0 */
    {
/* tunnel local IP, don't care if not for tunnel mode */
        (char*)0,
/* tunnel remote IP, don't care if not for tunnel mode */
        (char*)0,
/* policy rules */
        TM_PFLAG_BYPASS,
/* authentication algorithm */
        0,
/* encryption algorithm */
        0,
/* lifetime in seconds, if zero, use default value */
        0,
/* lifetime in kilo-bytes, if zero, use default value */
        0,
    },
/* policy content #1 */

```

```
        {
            "1.1.1.1",
            "2.2.2.1",
            TM_PFLAG_ESP + TM_PFLAG_TUNNEL + TM_PFLAG_PROTO_PACKET,
            SADB_AALG_MD5HMAC,
            SADB_EALG_3DESCBC,
            0,
            0
        },

/* policy content #2 */
        {
            (char*)0,
            (char*)0,
            TM_PFLAG_AH + TM_PFLAG_TRANSPORT + TM_PFLAG_PROTO_PACKET,
            SADB_AALG_SHA1HMAC,
            0,
            0,
            0
        }
    };

/* This is a VERY important table. This specifies which rules are
 * applied to received packets (TM_IPSEC_INBOUND), sent packets
 * (TM_IPSEC_OUTBOUND), AND IN WHAT ORDER.
 */
    ttIpssecPolicyPair plcyPair[] =
    {
        {0, 0, TM_IPSEC_BOTH_DIRECTION},
        {1, 1, TM_IPSEC_BOTH_DIRECTION},
        {1, 2, TM_IPSEC_BOTH_DIRECTION}
    };

/* In the last entry, selector index (1) is same as the previous entry,
 * that means they are bundled policy contents, they must worked
 * together to protect the traffic. The inner policy content (index =1,
 * the ESP tunnel one) will be applied first, followed by the outer
 * policy content (index = 2, the AH transport one)
 */
/* Also notice that selector index 2 is not used here. We are going to use it to
generate SA manually */

};

errorCode = tfPolicyRestore(plcyPair,
    plcySelector,
    plcyContent,
    3);

if(errorCode != TM_ENOERROR )
{
    return errorCode;
}

/*6.1.9.2 Manually add four SAs for UDP packet. User is responsible to add SAs
for TCP, ICMP packets. Each policy content needs two SAs (inbound + outboud), we
have bundle policy contents, so we need to add four SAs for protecting UDP
packets. */
```

```
/* Add one pair of SA according to the inner policy (policy content = 1) ,
inbound SPI = 1000, use keymaterial keyMat1; outbound SPI = 1100, use
keymaterial keyMat2. plcySelector[2] is specified selector for UDP packet */

    errorCode = tfSadbRecordManualAdd(&sadbPtr,
                                      TM_IPSEC_INBOUND,
                                      1000,
                                      &plcySelector[2],
                                      1, /* the ESP tunnel content*/
                                      keyMat1,
                                      44); /*3des + md5: we need at least 40
bytes */
    if(errorCode)          goto COMMON_RETURN;

    errorCode = tfSadbRecordManualAdd(&sadbPtr,
                                      TM_IPSEC_OUTBOUND,
                                      1100,
                                      &plcySelector[2],
                                      1,
                                      keyMat2,
                                      44);
    if(errorCode)          goto COMMON_RETURN;

/* Add one pair of SA according to the outer policy (policy content = 2) ,
inbound SPI = 2000, use keymaterial keyMat3; outbound SPI = 2100, use
keymaterial keyMat4. plcySelector[2] is specified selector for UDP packet */

    errorCode = tfSadbRecordManualAdd(&sadbPtr,
                                      TM_IPSEC_INBOUND,
                                      2000,
                                      &plcySelector[2],
                                      2,
                                      keyMat3,
                                      25); /*sha1, we need at least 20*/
    if(errorCode)          goto COMMON_RETURN;

    errorCode = tfSadbRecordManualAdd(&sadbPtr,
                                      TM_IPSEC_OUTBOUND,
                                      2100,
                                      &plcySelector[2],
                                      2,
                                      keyMat4,
                                      25);

COMMON_RETURN:

    return errorCode;
}
```

/*

6.1.10 Procedure testIpsecDifferentPolicy

*/

```

static int testIpsecDifferentPolicy(ttUserIpAddress clientIPAddr,
                                   ttUserIpAddress peerIPAddr,
                                   int iterations)
{
    ttIpsecSelector      selector;
    ttSadbRecordPtr     sadbPtr;
    ttPolicyEntryPtr    plcyPtr;
    int                 errorCode = 0;

    errorCode = tfIpsecPolicyQueryByIndex(1, &plcyPtr);
    if(errorCode != TM_ENOERROR) return errorCode;
    tm_bzero(&selector, sizeof(ttIpsecSelector));
/* UDP IPsec test*/
    printf("Case 1: UDP in IPsec, no IKE. Use the manually input SA \n");
    testUdp( clientIPAddr, peerIPAddr, iterations);

    selector.selLocIp1.ss_family = PF_INET;
    selector.selRemtIp1.ss_family = PF_INET;
    selector.selLocIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selRemtIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selLocIp1.addr.ipv4.sin_addr.s_addr = inet_addr("1.1.1.1");
    selector.selRemtIp1.addr.ipv4.sin_addr.s_addr = inet_addr("2.2.2.1");
    selector.selRemtPort = 0;
    selector.selLocPort = 0; /* not used anyway*/
    selector.selProtocol = TM_IP_UDP;
    selector.selIpFlags = TM_SELECTOR_BOTH_IP_HOST;

/* Delete the outer SA and test again, should drop the packet */
    tfSadbRecordFind(0, &selector, &sadbPtr, plcyPtr,
                    plcyPtr->plcyContentPtr->pctOuterContentPtr,
                    TM_IPSEC_OUTBOUND);

    if(sadbPtr)
    {
        errorCode = tfSadbRecordDelete(0, sadbPtr);
        if(errorCode != TM_ENOERROR) return errorCode;
    }

    printf("\nCase 2: UDP in IPsec, without outer outbound SA, drop the
packet\n");
    testUdp( clientIPAddr, peerIPAddr, iterations);

/* Delete the outbound Policy, will use the BYPASS policy, incoming policy check
* fails
*/

    tfPolicyDelete(plcyPtr, TM_IPSEC_OUTBOUND);

    printf("\nCase 3: UDP in IPsec, no outbound policy, will \
use the bypass policy.(Will fail in \
the inbound policy check ) \n");
    testUdp( clientIPAddr, peerIPAddr, iterations);

```

```
/* Delete the inbound policy, use BYPASS policy*/
    tfPolicyDelete(plcyPtr, TM_IPSEC_INBOUND);
    printf("\nCase 4: UDP in IPsec, no specific policy except\
        bypass, should pass\n");
    testUdp( clientIPAddr, peerIPAddr, iterations);

/* delete all policies, fail because no policy found */
    tfPolicyClear();
    printf("\nCase 5: No bypass policy, send-out error\n");
    testUdp(clientIPAddr, peerIPAddr, iterations);

    return TM_ENOERROR;
}

/* the testUdp is just a UDP send /receive example, user may refer to Treck User
 * Manual for details. It is not provided here.
 */
```

6.2 IKE (Automatic keying)

```
/*
```

6.2.1 Define the Policy

```
* Since IPSEC will be used, for any traffic(even the bypass one), we need to
* have its policy.
* Policies at 1.1.1.1, in the order of preference:
*
* Any protocol, any port traffic between 1.1.1.0/24 and 2.2.1.0/24, use ESP
* tunnel mode + AH transport mode to protect. ESP algorithm: 3DES-CBC+ MD5-
* HMAC - 96 and, AH algorithm: SHA1-HMAC-96. Tunneled between 1.1.1.1 to
* 2.2.2.1, use
* packet value for upper layer protocols.
* Any to Any, BYPASS IPsec.
*/
```

```
/*
```

6.2.2 Set trsystem.h

```
* Define TM_USE_IPSEC
* Define TM_USE_IKE
* And make sure you need PFS feature or not, are you going to use Aggressive
* mode or not. See section 3 for the settings of each IPsec|IKE related macros
*/
```

```
#include <stdio.h>
```

```
#include <trsecapi.h>
```

```
#define TM_BUF_ARRAY_SIZE      4000
```

```
char    bufferArray[TM_BUF_ARRAY_SIZE];
```

```
char    bufferReceive[TM_BUF_ARRAY_SIZE];
```

```
static void testUdp(ttUserIpAddress clientIpAddr,
                   ttUserIpAddress peerIpAddr,
                   int iterations);
```

```
static int  addPolicyAndPsk(void);
```

```
static int testIpsecDifferentPolicy(ttUserIpAddress clientIpAddr,
                                   ttUserIpAddress peerIpAddr,
                                   int iterations);
```

```
int main ( )
```

```
{
    ttUserLinkLayer linkLayerHandle;
    ttUserInterface interfaceHandle;
    ttUserIpAddress clientIPAddr;
    ttUserIpAddress peerIPAddr;
    ttUserIpAddress mask;
    int              configFlags;
    int              scatteredBufferCount;
    int              errorCode;
    int              iterations;
```

```

/* Allow scatter send */
    configFlags = TM_DEV_SCATTER_SEND_ENB;
    iterations = 100;
    clientIPAddr = inet_addr("1.1.1.1");
    peerIPAddr = inet_addr("2.2.2.1");
    mask = inet_addr("255.255.255.0");

    if (configFlags & TM_DEV_SCATTER_SEND_ENB)
    {
        scatteredBufferCount = 1501;
    }
    else
    {
        scatteredBufferCount = 1;
    }
}

```

```

/*

```

6.2.3 Start Treck

```

*/
    errorCode = tfStartTreck();
    if (errorCode != TM_ENOERROR)
    {
        printf( "tfStartTreck failed '%s'\n", tfStrError(errorCode));
        return errorCode;
    }

```

```

/*

```

6.2.4 Add driver interface below Ethernet link layer

```

*/
    linkLayerHandle = tfUseEthernet();
    interfaceHandle = tfUseLinuxDriver("TEST", linkLayerHandle, &errorCode);

    if (errorCode != TM_ENOERROR)
    {
        printf( "Adding interface driver failed '%s'",
                tfStrError(errorCode) );
        return errorCode;
    }

```

```

/*

```

6.2.5 Call tfUseIpsec

Initialize IPsec global variables

```

*/
    errorCode = tfUseIpsec();
    if (errorCode != TM_ENOERROR)
    {
        printf( "tfUseIpsec failed '%s'\n");
        return errorCode;
    }

```

```

/*

```

6.2.6 Add policy

For IKE, you don't need to add SA

```
*/
    errorCode = addPolicy();
    if (errorCode != TM_ENOERROR)
    {
        printf( "addPolicy failed \n");
        return errorCode;
    }

```

```
/*
```

6.2.7 Open the added interface

```
*/
    errorCode = tfOpenInterface( interfaceHandle,
                                clientIPAddr,
                                mask,
                                configFlags,
                                scatteredBufferCount );

    if (errorCode != TM_ENOERROR)
    {
        printf("tfConfigInterface failed '%s'\n", tfStrError(errorCode));
        return errorCode;
    }

```

```
/*
```

6.2.8 Call tfStartIke

start IKE

```
*/
    errorCode = tfStartIke(TM_DOI_ID_IPV4_ADDR, 4, (void*)&clientIPAddr);
    if (errorCode != TM_ENOERROR)
    {
        printf( "tfUseIpsec failed '%s'\n");
        return errorCode;
    }

```

```
/*
```

6.2.9 Add Preshared-key

For IKE, you don't need to add SA

```
*/
    errorCode = tfPresharedKeyAdd("2.2.2.1", "presharedKey#1", 0);
    if (errorCode != TM_ENOERROR)
    {
        printf( "add preshared key failed \n");
        return errorCode;
    }

```

```
/*
```

6.2.10 Test case

Run different policy combinations

```
*/
    testIpsecDifferentPolicy(clientIPAddr, peerIPAddr, iterations);

    return errorCode;

```

```

}

/*
6.2.11 Procedure addPolicy
*/

static int addPolicy (void)
{
    int                errorCode = 0;

    ttIpsecSelectorInString plcySelector[] =
    {
/* selector #0, ANY to ANY */
        {
            (char*)0,                0,
            (char*)0,                0,
            (ttl6Bit)0 ,
            TM_SELECTOR_WILD_PORT,    TM_SELECTOR_WILD_PORT,
            TM_SELECTOR_WILD_PROTOCOL,
        },
/* selector #1, From 1.1.1.0/24 to 2.2.2.1, any port, any protocol */
        {
            "1.1.1.0",                int)"255.255.255.0",
            "2.2.2.1",                0,
            TM_SELECTOR_LOCIP_SUBNET + TM_SELECTOR_REMTIP_HOST,
            TM_SELECTOR_WILD_PORT,    TM_SELECTOR_WILD_PORT,
            TM_SELECTOR_WILD_PROTOCOL,
        }
    };

    ttPolicyContentInString plcyContent[] =
    {
/* policy content #0 */
        {
            (char*)0,
            (char*)0,
            TM_PFLAG_BYPASS,
            0,
            0,
            0,
            0
        },
/* policy content #1 */
        {
            "1.1.1.1",
            "2.2.2.1",
            TM_PFLAG_ESP + TM_PFLAG_TUNNEL + TM_PFLAG_PROTO_PACKET,
            SADB_AALG_MD5HMAC,
            SADB_EALG_3DESCBC,
            0,
            0
        },
/* policy content #2 */
        {

```

```
        (char*)0,  
        (char*)0,  
        TM_PFLAG_AH + TM_PFLAG_TRANSPORT + TM_PFLAG_PROTO_PACKET,  
        SADB_AALG_SHA1HMAC,  
        0,  
        0,  
        0  
    }  
};
```

```
ttIpsecPolicyPair plcyPair[] =  
{  
    {0, 0, TM_IPSEC_BOTH_DIRECTION},  
    {1, 1, TM_IPSEC_BOTH_DIRECTION},  
    {1, 2, TM_IPSEC_BOTH_DIRECTION}  
};  
  
errorCode = tfPolicyRestore(plcyPair,  
    plcySelector,  
    plcyContent,  
    3);  
  
if(errorCode != TM_ENOERROR )  
{  
    return errorCode;  
}
```

COMMON_RETURN:

```
    return errorCode;  
}
```

```

/*
6.2.12 Procedure testIpsecDifferentPolicy
*/

static int testIpsecDifferentPolicy(ttUserIpAddress clientIPAddr,
                                   ttUserIpAddress peerIPAddr,
                                   int iterations)
{
    ttIpsecSelector      selector;
    ttSadbRecordPtr      sadbPtr;
    ttPolicyEntryPtr     plcyPtr;
    int                  errorCode = 0;

    errorCode = tfIpsecPolicyQueryByIndex(1, &plcyPtr);
    if(errorCode != TM_ENOERROR) return errorCode;
    tm_bzero(&selector, sizeof(selector));
/* UDP IPsec test*/
    printf("Case 1: UDP in IPsec, Should trigger IKE and after IKE, send the
packets \n");
    testUdp( clientIPAddr, peerIPAddr, iterations);

    selector.selLocIp1.ss_family = PF_INET;
    selector.selRemtIp1.ss_family = PF_INET;
    selector.selLocIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selRemtIp1.ss_len = sizeof(struct sockaddr_in);
    selector.selLocIp1.addr.ipv4.sin_addr.s_addr = inet_addr("1.1.1.1");
    selector.selRemtIp1.addr.ipv4.sin_addr.s_addr = inet_addr("2.2.2.1");
    selector.selRemtPort = 0;
    selector.selLocPort = 0; /* not used anyway*/
    selector.selProtocol = TM_IP_UDP;
    selector.selIpFlags = TM_SELECTOR_BOTH_IP_HOST;

/* Delete the outer SA and test again, should trigger IKE again to negotiate an
* outer SA
*/
    tfSadbRecordFind(0, &selector, &sadbPtr, plcyPtr,
                    plcyPtr->plcyContentPtr->pctOuterContentPtr,
                    TM_IPSEC_OUTBOUND);

    if(sadbPtr)
    {
        errorCode = tfSadbRecordDelete(0, sadbPtr);
        if(errorCode != TM_ENOERROR) return errorCode;
    }

    printf("\nCase 2: UDP in IPsec, without outer outbound SA, trigger IKE again
\n");
    testUdp( clientIPAddr, peerIPAddr, iterations);

/* Delete the outbound Policy, will use the BYPASS policy, incoming policy check
* fails
*/
    tfPolicyDelete(plcyPtr, TM_IPSEC_OUTBOUND);

```

```
    printf("\nCase 3: UDP in IPsec, no outbound policy, will \\  
        use the bypass policy.(Will fail in \\  
        the inbound policy check ) \n");  
    testUdp( clientIPAddr, peerIPAddr, iterations);  
  
/* Delete the inbound policy, use BYPASS policy*/  
    tfPolicyDelete(plcyPtr, TM_IPSEC_INBOUND);  
    printf("\nCase 4: UDP in IPsec, no specific policy except\  
        bypass, should pass\n");  
    testUdp( clientIPAddr, peerIPAddr, iterations);  
  
/* delete all policies, fail because no policy found */  
    tfPolicyClear();  
    printf("\nCase 5: No bypass policy, send-out error\n");  
    testUdp(clientIPAddr, peerIPAddr, iterations);  
  
    return TM_ENOERROR;  
}  
/* the testUdp is just a UDP send /receive example, user may refer to Treck User  
 * Manual for details. It is not provided here.  
 */
```

7 References

The Treck IPsec and IKE refers to the following documents:

Overview RFCs

- 2401 Security Architecture for the Internet Protocol
- 2411 IP Security Document Roadmap

Basic protocols

- 2402 IP Authentication Header
- 2406 IP Encapsulating Security Payload (ESP)

Key management

- 2407 The Internet IP Security Domain of Interpretation for ISAKMP
- 2408 Internet Security Association and Key Management Protocol (ISAKMP)
- 2409 The Internet Key Exchange (IKE)
- 2412 The OAKLEY Key Determination Protocol
- 2631 Diffie-Hellman Key Agreement Method

Details of various items used

- 1321 The MD5 Message-Digest Algorithm.
- 1828 IP Authentication using Keyed MD5
- 1829 The ESP DES-CBC Transform
- 1851 The ESP Triple DES Transform
- 1852 IP Authentication using Keyed SHA.
- 2085 HMAC-MD5 IP Authentication with Replay Prevention
- 2104 HMAC: Keyed-Hashing for Message Authentication
- 2144 The CAST-128 Encryption Algorithm
- 2202 Test Cases for HMAC-MD5 and HMAC-SHA-1
- 2403 The Use of HMAC-MD5-96 within ESP and AH
- 2404 The Use of HMAC-SHA-1-96 within ESP and AH
- 2405 The ESP DES-CBC Cipher Algorithm With Explicit IV
- 2410 The NULL Encryption Algorithm and Its Use With IPsec
- 2437 PKCS #1: RSA Cryptography Specifications Version 2.0
- 2451 The ESP CBC-Mode Cipher Algorithms
- 2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- 2857 The Use of HMAC-RIPED-160-96 within ESP and AH
- 3174 US Secure Hash Algorithm 1 (SHA1)
- FIPS (Federal Information Processing Standards) Publication 197 - Advanced Encryption Standards (AES)
- B. Schneier, etc, Twofish: A 128-Bit Block Cipher
- B. Schneier, Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)

Related Documents

- draft-ietf-ipsec-ikev2-02.txt
- draft-jenkins-ipsec-rekeying-06.txt
- draft-spencer-ipsec-ike-implementation.htm

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
All rights reserved.

This product includes software written by Tim Hudson (tjh@cryptsoft.com)

