An architecture for the Internet Key Exchange Protocol

by P.-C. Cheng

In this paper we present the design, rationale, and implementation of the Internet Key Exchange (IKE) Protocol. This protocol is used to create and maintain Internet Protocol Security (IPSec) associations and secure tunnels in the IP layer. Secure tunnels are used to construct virtual private networks (VPNs) over the Internet. The implementation is done in the application layer. The design includes four components: (1) an IKE protocol engine to execute the IKE protocol, (2) a tunnel manager to create and manage secure tunnels—it generates requests to the IKE protocol engine to establish security associations, (3) VPN policy administration tools to manage VPN policies that guide the actions of the IKE protocol engine and the tunnel manager, and (4) a certificate proxy server to acquire and verify public key certificates that are used for authentication of messages and identities in the IKE protocol. The implementation was done on the Advanced Interactive Executive® (AIX®) operating system at IBM Research and has been transferred to IBM's AIX, Application System/400®, and System/390® products.

This paper is a follow-on to an earlier paper[1] and describes a design and its implementation of the Internet Key Exchange, or IKE, Protocol specified in the report by Harkins and Carrel.[2] The work was done from 1995 to 1999, during which period the IKE protocol was defined and then evolved in the Internet Engineering Task Force (IETF) Internet Protocol Security (IPSec) working group.[3] Although the protocol is still evolving, it has become stable enough for participants of the working group to build independent and interoperable implementations. This paper describes IBM's implementation. The work was done on the IBM Advanced Interactive Executive* (AIX*) version 4 operating system at IBM Research, and the core technology has been transferred to IBM product divisions producing AIX, AS/400* (Application System/400*), and S/390* (System/390*), which have since augmented the core technology and made it part of the product offerings.

IKE is the primary protocol to generate and maintain IPSec[1,4] security associations, which are the basic building blocks of virtual private networks (VPNs) over the Internet. IKE uses cryptography[5,6] extensively, and this paper assumes that readers have a basic knowledge of cryptography. Readers should be familiar with concepts such as symmetric key cryptography, public key cryptography, encryption, hash, Diffie-Hellman Key Agreement, public key signature/encryption, public key certificates, certificate authority, and public key infrastructure.

IKE is a complex protocol, and a detailed description of it is outside the scope of this paper. We only introduce it with enough details to put the description of our design and implementation in context. The introduction is nonetheless lengthy because of the complexity of the protocol.

In the remainder of the paper, the next section is an introduction to the protocol, the subsequent section describes our architecture and implementation of the

protocol, and the last section gives a brief description of the performance of our implementation.

The protocols

The Internet Key Exchange Protocol is fairly complex, and interested readers should refer to References 2, 7, and 8 for details.

IKE is specified by the Internet Society RFC (Request for Comments) 2409[2] that references the Internet Security Association and Key Management Protocol (ISAKMP)[7] and The Internet IP Security Domain of Interpretation (DOI) for ISAKMP.[8] ISAKMP specifies the high-level, abstract syntax and semantics for certain types of key management protocols. Thus, while the letter "P" in the abbreviation "ISAKMP" means "protocol," ISAKMP specifies only a *framework* for key management protocols but not any implementable protocol since the specification lacks sufficient low-level details. IKE and DOI fill in the details and specify a set of implementable protocols that fit into the framework. Whereas IKE focuses mainly on the detailed protocol semantics, DOI focuses mainly on the detailed syntax and semantics of the information carried by the messages of the protocol. In ISAKMP terminology, a protocol fitting into its framework is called a *key exchange protocol*. A protocol under the ISAKMP framework should use User Datagram Protocol (UDP)[9] port 500 to send and receive messages.

Note that one can design an IKE-like protocol without referring to a framework. So one has to ask why ISAKMP is needed and what it provides. In the rest of this section we first discuss ISAKMP and DOI in more detail and then discuss IKE in more detail.

ISAKMP. ISAKMP is a definition of a high-level, abstract framework for point-to-point, two-party, asymmetric key management protocols. Being asymmetric means the two parties have different roles. One role is called the *initiator*, which begins the exchange of protocol messages by sending the first message; the other role is called the *responder*, which replies to the first message from the initiator.

Why ISAKMP? ISAKMP makes a distinction between "key exchange" and "key management" and considers the latter to be a superset of the former.

Key exchange is mainly concerned with exchanging information to generate secret keys shared between two parties. ISAKMP requires a key exchange protocol to:

- Generate a set of secret key(s) shared exclusively between the two parties
- Authenticate the identity of each party to the other. Here, "authenticating identity" means authenticating the binding between a party's claimed identity and the pieces of information the party claims to have sent and received. In particular, a successful authentication should prove the freshness of the information to defeat replay.
- Ensure the set of secret keys generated by one protocol message exchange is independent of key sets generated by other protocol message exchanges. This means compromise of one key set does not lead to compromise of other sets. This property is usually known as *perfect forward secrecy* (PFS).[10]
- Be scalable. Here scalability means that a key exchange protocol can be executed between any two parties within a very large population, even if the two parties do not share any secret *a priori*. This requirement, coupled with the requirement for authentication, implies the use of public key cryptography[5,6,11] and dependency on the public key infrastructure (PKIX),[12,13] a very complex topic in itself. The process of acquiring and verifying public key certificates is outside the scope of ISAKMP and IKE specification, although an ISAKMP/IKE implementation must have the means to do so. More discussion on PKIX and ISAKMP is given in the next subsection.

A shared set of secret keys would be of little value unless the two parties also agree on some meta parameters. This agreement should include information on:

- How to use the keys:
  - –Cryptographic algorithm(s) to be used with the keys
  - –Parameters for the cryptographic algorithm(s), such as key size and initial synchronization
  - –How and to what the cryptographic algorithm(s) and the keys should be applied
- Key lifetime and key refreshment policy[1]
- An identifier for the meta parameters and the keys. The identifier should be unique with respect to the two parties.

These meta parameters, the shared keys, the identities, and the IP addresses of the two parties form a security association, or SA,[1,4] and ISAKMP key management is concerned with creating and managing

security associations exclusively shared between two parties. Therefore, a key management protocol that fits into the ISAKMP framework is logically divided into the following two steps:

1. Parameter negotiation: To agree on the meta parameters and a key exchange protocol and its parameters

   To carry out this step, the initiator sends a list of proposals to the responder in the first message. The responder either chooses one and only one proposal from the list and sends its choice to the initiator or rejects the entire list and sends back an error in the second message.

   What to propose and what to choose depends on the two parties' security policies. A more detailed description of an ISAKMP proposal and the parameter negotiation is given in a later subsection.

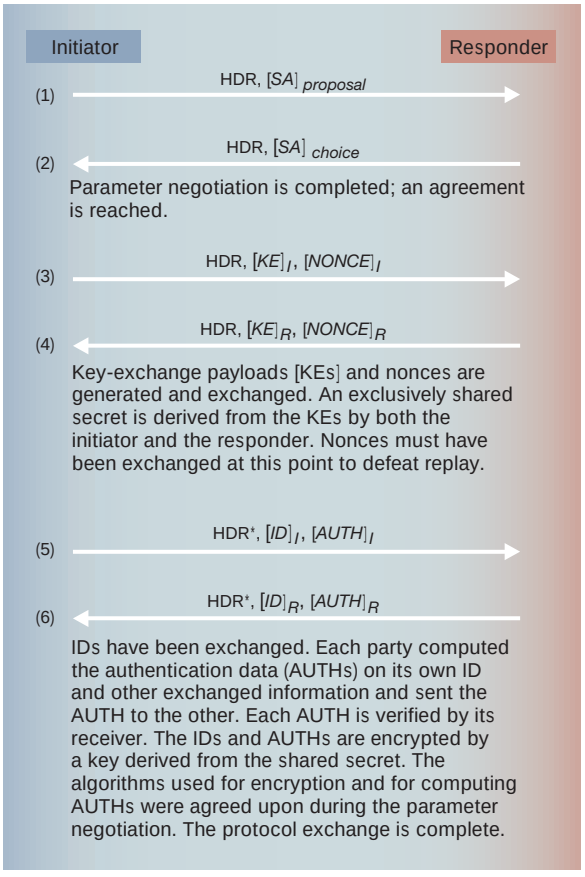
2. Key exchange: To execute the agreed-upon key exchange protocol to generate keys and to authenticate each party to the other

In ISAKMP terminology, the two parties execute the two steps to negotiate SAs. An instance of the execution of the two steps is called a *negotiation*. From now on, the word "negotiation" refers to such an instance unless "parameter negotiation" is used.

This two-step division allows ISAKMP to separate key generation, which depends heavily on a specific key exchange protocol, from security association management, which could be conducted in a generic way independent of the key exchange protocol. It also allows more than one key exchange protocol to fit into the framework.

ISAKMP has a well-defined encoding format for the list of proposals. In theory, the ISAKMP framework specification could stop here and allow any key exchange protocol meeting its requirements to fit into the framework. However, researchers have shown[10,14–16] that designing and analyzing a key exchange protocol is very difficult. Subtle mistakes can render a key exchange protocol vulnerable to attacks even if the protocol uses only strong cryptography primitives. For this reason, ISAKMP specification includes a few rigid templates, called *exchanges*, for key exchange protocols. The belief is that a key exchange protocol that fits into a template should meet the ISAKMP requirements and thus be "secure." A template specifies: the number of messages sent by each party, the ordering of messages, the types of information a message should carry (ISAKMP defines different types of payloads to carry different types of information), and the actions, together with their effects, a party should take when sending or receiving a particular message.

Figure 1 ISAKMP identity-protection exchange



Thus, a template specification allows a message-by-message check against a protocol. Figure 1 depicts the ISAKMP identity-protection exchange. Messages 1 and 2 carry out the parameter negotiation; messages 3 to 6 carry out the key exchange. HDR is the ISAKMP message header, which contains anticlogging cookies (see later subsection), protocol version, message length, etc. Each ISAKMP payload is shown in square brackets. The SA payloads contain the initiator's proposals and the responder's choice. The [*KE*]s are the key exchange payloads used to derive a shared secret. [*NONCE*]s are nonces, large and

Figure 2 ISAKMP aggressive exchange

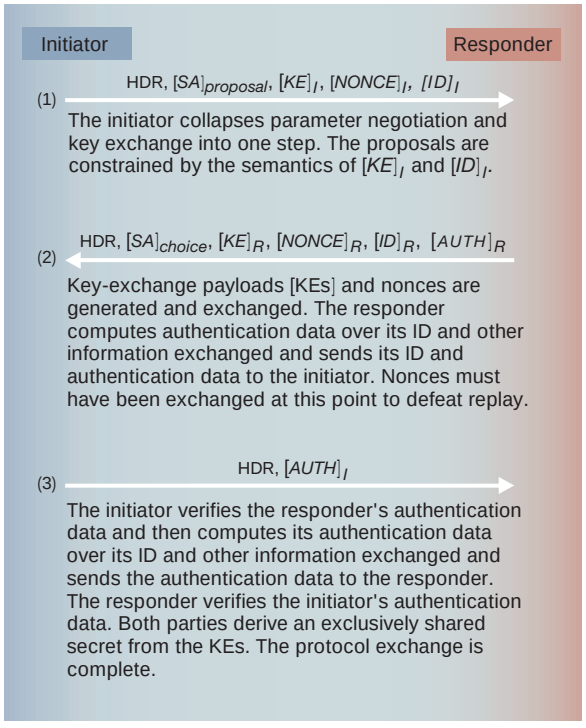
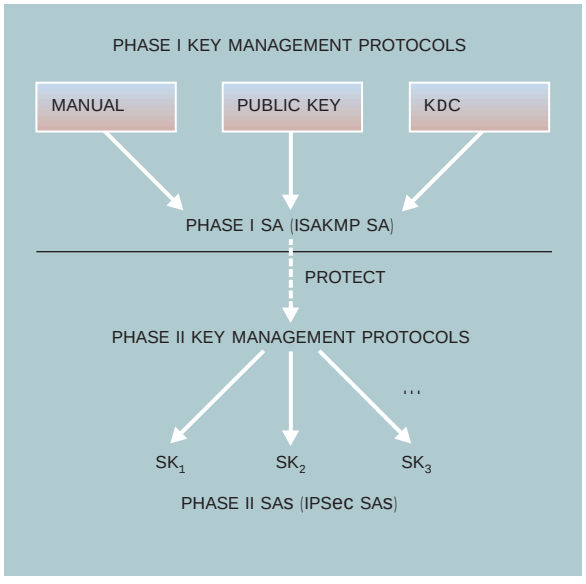


Figure 3 Two-phase approach to key management



never-used-before random numbers, used to defeat replay. The terms $[ID]_I$ and $[ID]_R$ are the identities of the initiator and the responder, respectively. In messages 5 and 6, there is a "*" symbol following HDR, meaning that all payloads following the message header are encrypted. Therefore, the identities are encrypted, and thus the name "identity-protection." [*AUTH*]s contain data used for authentication.

Figure 2 depicts the ISAKMP aggressive exchange. The aggressive exchange collapses the identity-protection exchange into three messages by conducting the parameter negotiation and the key exchange at the same time. The price paid for fewer messages is that identities generally cannot be encrypted and the room for negotiation is constrained. For example, the key exchange protocol cannot be negotiated, so the initiator chooses a key exchange protocol and indicates the choice in all the proposals. The responder can only accept a proposal or reject all proposals. More details on constraints on proposals will be discussed in the subsection on IKE Phase I protocols.

The two-phase approach. Another important concept in ISAKMP is the two-phase approach, which is orthogonal to other concepts discussed so far. This concept was first discussed in References 17 and 18 and then in Reference 7 and Reference 1. Figure 3 depicts this concept. This concept is not strictly needed to create security associations, but it can be argued that for most application scenarios this concept has significant methodological and design value.[1] The two-phase approach is meant to address the following two problems faced by key management protocols:

1. How to share authenticated information (SAs), including secrets, identities, etc., between two parties when no secret is shared *a priori*
2. How to efficiently refresh the shared information to defeat cryptography analysis

The first problem is solved by public key cryptography; IKE is a good example of such a solution. The second problem emphasizes efficiency and thus conflicts with the use of public key cryptography because its computation cost is usually very high. The two-phase approach resolves the conflict as follows.

In Phase I, use public key cryptography and run a key management protocol *infrequently* to generate the "first" shared security association. This SA is called an "ISAKMP SA" or a "Phase I SA." The secret keys in this SA are associated with symmetric key

cryptographic algorithms that are much more efficient than those of public key cryptography. These keys and algorithms are used to protect Phase II protocols.

In Phase II, under the protection of an ISAKMP SA, run a key management protocol *frequently* to generate many more SAs. These Phase II SAs are used to protect data (i.e., not key management) traffic between the two parties. Experience[18] has shown that protocols used in Phase II can be very efficient.

In theory, protocols fitting the ISAKMP "exchanges" can be used in both phases. But since Phase II negotiations are protected by ISAKMP SAs, more efficient protocols can usually be designed for Phase II negotiations. This is the case for the IKE Phase II protocol described in more detail later.

ISAKMP proxy negotiation. ISAKMP proxy negotiation means an initiator and a responder, already sharing a Phase I ISAKMP SA, use this ISAKMP SA to negotiate Phase II SAs for protecting traffic between other systems. This situation is best illustrated by the example in Figure 4.

In Figure 4, two firewalls, FW-1 and FW-2, share an ISAKMP SA and use this ISAKMP SA to negotiate IPSec SAs to protect communication between A and B. Note that in this case the end points of these IPSec SAs are still FW-1 and FW-2, but they will be designated to protect the communication between A and B only. This designation happens when the identities of A and B, instead of those of FW-1 and FW-2, are exchanged during a Phase II negotiation. The identities of A and B are called ID_{ui} and ID_{ur}.[7]

Either A or B can be a single system or a group of systems (see the subsection on DOI). It is also possible that a single system A coincides with FW-1, in which case FW-1 disappears and A acts as a proxy of itself.

The ISAKMP proposal and parameter negotiation. Figure 5 shows the logical structure of an ISAKMP proposal. It is a three-layer hierarchy consisting of:

1. Security protocols—the functionality of the to-be-generated SAs. For example, the security protocol for ISAKMP Phase I negotiation is always "ISAKMP," which indicates the Phase I SA is to protect Phase II messages. For Phase II protocols generating IPSec SAs, the security protocol can be ESP[19] or AH.[20]

Figure 4 Firewalls as ISAKMP proxies for systems behind them

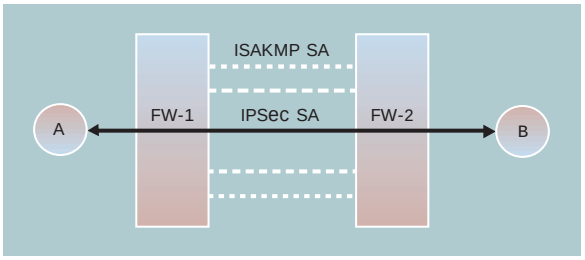
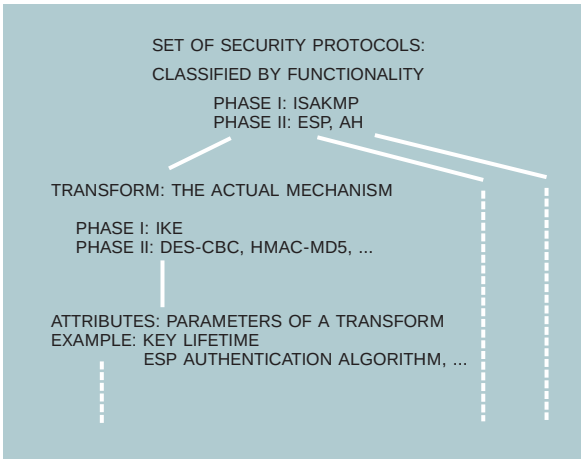


Figure 5 Structure of an ISAKMP proposal

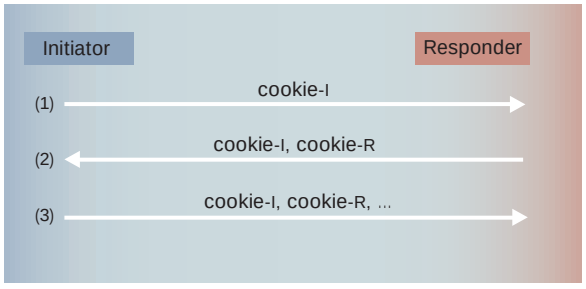


Each security protocol proposes meta parameters of a to-be-generated SA. If more than one protocol is in a proposal, then all these to-be-generated SAs should be used and managed together. For example, if both ESP and AH appear in a proposal, the resulting ESP SA and AH SA must be placed in an SA bundle.[1]

2. Transform—indicate the actual mechanism to provide the functionality indicated by the security protocol. A security protocol proposed by the initiator can have a list of transforms arranged in descending order of preference. The responder must either choose one and only one transform or reject the entire list.

   In a Phase I negotiation, a transform names a key exchange protocol such as IKE. In a Phase II negotiation generating IPSec SAs, transform names the actual cryptographic algorithm to achieve the

Figure 6 Phil Karn's anticlogging cookie



functionality. For example, a transform under ESP can be DES (Data Encryption Standard), triple-DES, or other encryption algorithms; a transform under AH is usually HMAC-SHA[21] or HMAC-MD5.[22]

3. Transform attributes—include in each transform a set of attributes from the initiator. For a Phase I negotiation, these attributes can be applied to the key exchange protocol or to the to-be-generated ISAKMP SA, such as the lifetime of the SA, or to both. For a Phase II negotiation generating IPSec SAs, these attributes can be SA lifetime, authentication algorithm for ESP, key size for cryptographic algorithms with variable-size keys, etc.

In general, a proposal from the initiator can be thought of as a product-of-sums Boolean formula, with each sum being an ordered list of transforms of a security protocol in descending order of preference. For example, the following formula ("·":AND, "|":OR) represents a proposal for an ESP/AH SA bundle:

$$\overbrace{(Triple\text{-}DES|DES)}^{ESP} \cdot \underbrace{(HMAC\text{-}SHA|HMAC\text{-}MD5)}_{AH} \quad (1)$$

Expanding Expression 1, we get the sum-of-products form:

$$\begin{aligned} &Triple\text{-}DES \cdot HMAC\text{-}SHA| \\ &Triple\text{-}DES \cdot HMAC\text{-}MD5| \\ &DES \cdot HMAC\text{-}SHA| \\ &DES \cdot HMAC\text{-}MD5 \end{aligned} \quad (2)$$

The responder must either accept the proposal by choosing *one and only one* item from Expression 2 or reject the entire proposal.

Each proposal is encoded in a set of ISAKMP PROPOSAL payloads; each PROPOSAL payload is for one security protocol in the proposal and contains the identifier of the security protocol and the encoding of the list of transforms for the security protocol. Each transform and its attributes are encoded in an ISAKMP TRANSFORM payload. Payloads of the same proposal but different security protocols share the same proposal number in the header of the PROPOSAL payloads. A proposal list from the initiator is an array of sets of PROPOSAL payloads; the sets are arranged in descending order of preference and are placed in an SA payload. The responder's choice is encoded as one set of PROPOSAL payloads; each such payload corresponds to a security protocol and contains exactly one TRANSFORM payload.

In the case of IKE, a Phase II negotiation actually generates two unidirectional SAs[1,4] for each ⟨security protocol, transform⟩ chosen by the responder. One SA is to protect data communication from the initiator to the responder, and the other is to protect data communication from the responder to the initiator. For each PROPOSAL payload sent by the initiator in a Phase II negotiation, the initiator generates an identifier for the to-be-generated responder-to-initiator SA (see earlier subsection on the two-phase approach), called a *security parameter index* (SPI),[4] and places the SPI inside the header of the PROPOSAL payload. Likewise, the responder generates an SPI for each PROPOSAL payload it sends back. A responder's SPI identifies an initiator-to-responder SA.

The ISAKMP anticlogging cookies. Anticlogging cookies were first proposed by Phil Karn[23] to counter denial-of-service (DOS) attacks when the adversary is not a man-in-the-middle, active attacker. In the case of ISAKMP, an attacker starts a DOS attack by impersonating many different initiators to start many negotiations with a responder. The goal of the attack is to force the responder to waste valuable computing and network resources on useless negotiations.

Since it is generally impossible to prevent an attacker on the Internet from mounting such an attack, the goal of the anticlogging cookie is not to prevent DOS attacks from happening, but to reduce the amount of resources consumed by the attacks. The basic idea is that the responder will demand confirmation of

the genuineness of a negotiation from the supposed initiator before committing any significant amount of resources. Karn's original design is sketched in Figure 6, which shows the following actions:

1. The initiator generates a random number, cookie-I, such that ⟨responder, cookie-I⟩ is unique with respect to the initiator, and sends the cookie-I in the first message to the responder. Usually, cookie-I is generated from some local secret, some unique information about the responder (e.g., IP address), and possibly other local state information.

2. The responder generates a request for confirmation, ⟨cookie-I, cookie-R⟩, and sends the request to the supposed initiator. The cookie-R is a random number generated from cookie-I, some local secret, some unique information about the initiator (e.g., IP address), and possibly other local state information. The cookie-R must have the following properties:

   - The mapping from ⟨initiator, cookie-I⟩ to cookie-R is one-to-one with respect to the responder. This property implies that the responder always generates the same cookie-R from the same ⟨initiator, cookie-I⟩.
   - Only the responder can generate the cookie-R. This property comes from using a local secret in generating the cookie-R.

   These properties eliminate the need for the responder to remember cookie-I or cookie-R; therefore, the responder does not keep any state information on the first and second messages.

3. The initiator includes ⟨cookie-I, cookie-R⟩ as the requested confirmation in the third message. The responder can compute a new cookie-R from ⟨initiator, cookie-I⟩ and compare this new cookie-R with the one in the third message. If these two cookie-Rs match, then the responder can have some assurance that it is the supposed initiator, not an attacker, who sent the first message. If the supposed initiator did not send the first message, then it simply drops the second message; there is no need for the responder to do anything in this case since it does not keep state information on the first and second messages.

The generation of cookies must be efficient. Efficiency can usually be achieved with keyed-hash[24] or pseudorandom functions.[25] Therefore, the responder can efficiently recompute and verify the cookie-R in the third message without doing much computation.

The anticlogging cookie is only effective when the attacker is *not* an active man-in-the-middle. Otherwise, an active man-in-the-middle can always intercept the second message and sends ⟨cookie-I, cookie-R⟩ in the third message.

In ISAKMP, cookies are eight bytes long and are placed in the first two fields of an ISAKMP message header. In a Phase I negotiation, cookie-I is sent in the first message, and the responder sends back ⟨cookie-I, cookie-R⟩ in the second message; all following messages will contain this cookie pair. Thus, the responder in an ISAKMP negotiation does demand confirmation from the initiator; however, since the responder has to perform the parameter negotiation and has to maintain the negotiation result, the anticlogging effect is reduced.

Because of the uniqueness of the cookies, ⟨cookie-I, cookie-R⟩ generated during a particular ISAKMP Phase I negotiation is used as the identifier of the ISAKMP SA generated by the negotiation. The cookie pair can also be used to identify messages of a particular Phase I negotiation.

For messages of a Phase II negotiation under the protection of an ISAKMP SA, the message headers always carry the ⟨cookie-I, cookie-R⟩ of the SA. Another field in the message header, called *message ID*, is used to identify messages of a particular Phase II negotiation; this field is zero for messages in a Phase I negotiation. The initiator of a Phase II negotiation will assign a unique random number to the message ID field, and all messages of the negotiation will carry this random number in the field.

DOI. In ISAKMP, a domain of interpretation (DOI) for ISAKMP is the context in which a key management protocol operates. It defines the syntax and semantics of all information that is relevant to the operation of the protocol. ISAKMP allows more than one DOI, but a key management protocol must indicate the DOI in which it is operating by filling the DOI field in the SA payload.

In practice, a DOI defines the syntax and semantics of the following:

- Everything being put in a proposal—These are identifiers for security protocols, transforms and attributes of each transform, and key exchange pro-

tocols. Valid values and encoding formats for each transform attribute are also defined. Parameters of a key exchange protocol are defined by the specification of that protocol.

- Identity types—Each type is assigned an identifier. Valid values and encoding formats for each type are also defined. Twelve identity types are defined in Reference 8, including a reserved NULL type. The types are listed below; values in parentheses are the identifiers of the types.

  –RESERVED (0)
  –ID_IPV4_ADDR (1): Four-byte Internet Protocol version 4 (IPv4, the version of Internet Protocol being used today)[26] address
  –ID_FQDN (2): Fully qualified Internet domain names,[27] such as foo.ibm.com
  –ID_USER_FQDN (3): Fully qualified Internet user domain names, that is, an e-mail address, such as joe@us.ibm.com
  –ID_IPV4_ADDR_SUBNET (4): An IPv4 address subnet,[28] encoded as a four-byte IPv4 address prefix followed by a four-byte mask
  –ID_IPV6_ADDR (5): A 16-byte Internet Protocol version 6 (IPv6, the version of Internet Protocol which is supposed to replace IPv4)[29] address
  –ID_IPV6_ADDR_SUBNET (6): An IPv6 address subnet, encoded as a 16-byte IPv6 address prefix followed by a 16-byte mask
  –ID_IPV4_ADDR_RANGE (7): A range of IPv4 addresses, encoded as two IPv4 addresses treated as unsigned integers, interpreted as the beginning and end of the range
  –ID_IPV6_ADDR_RANGE (8): A range of IPv6 addresses, encoded as two IPv6 addresses treated as unsigned integers, interpreted as the beginning and end of the range
  –ID_DER_ASN1_DN (9): ISO/ITU (International Organization for Standardization/International Telecommunication Union) X.500[30] distinguished name
  –ID_DER_ASN1_GN (10): X.500 GeneralName.[12] The X.500 GeneralName is for representing Internet identities, such as IP addresses, domain names, e-mail addresses, URIs,[31] etc., in an X.500 context.
  –ID_KEY_ID (11): An opaque byte stream identifying a secret key preshared between the initiator and the responder. This type of identity may be used when IKE is using an exclusively preshared key for authentication (see subsection on IKE Phase I protocols). The exact format of this type is vendor-specific.

  Among these identity types, subnets and address ranges identify a group of systems and should only be used in Phase II proxy negotiations. Other types identify individual systems (IP addresses and FQDNs, or fully qualified domain names), a user (USER_FQDN), or either (X.500 names), and can be used in both Phase I and Phase II negotiations. In a Phase II negotiation, if an identity used by one party cannot be resolved to an IP address, a subnet, or an address range, then it is assumed that the party is acting as a proxy of itself.

  An identity can be augmented by an Internet transport protocol number and an optional port number. Such augmentation can only be used in Phase II proxy negotiation. For example, using "9.2.253.6: TCP" means that the to-be-generated SAs are to protect TCP[32] traffic to and from the system with IP address 9.2.253.6; using "9.2.253.6:TCP:23" means to protect TELNET[33] traffic to and from the system with IP address 9.2.253.6. "23" is the well-known port number for the TELNET protocol. Therefore, with specific transport protocol and port number, an identity used in Phase II negotiation can identify a specific service on a system or on a group of systems.

- A special piece of information, called a *situation* in ISAKMP—Conceptually, a situation describes the characteristics of the specific communication a particular ISAKMP negotiation aims to protect. An SA payload has a field to hold these characteristics. "Situation" is supposed to aid the responder in deciding which proposal to pick. ISAKMP does not define any details for a situation but leaves the definition to the DOI.

  The Internet Security DOI[8] defines three kinds of "situation." One is "IDENTITY_ONLY." It is basically a null definition. It means a responder should use the supposed identities of both parties to make decisions during a parameter negotiation. The others are "SECRECY" and "INTEGRITY." They are defined according to the multilevel security concept in Reference 34. They are optional and are not implemented in our code.

IKE. IKE is a family of key exchange protocols defined according to the ISAKMP framework. The design of IKE is influenced by STS (Station-to-Station) Protocol,[10] SKEME,[14] and OAKLEY.[35] We discuss the IKE Phase I protocols first and then the IKE Phase II protocol.

IKE Phase I protocols. The Phase I protocols of IKE are based on the ISAKMP identity-protection and aggressive exchanges. The Internet Society RFC 2409 uses the word "mode" instead of "exchange"; *IKE main mode* refers to the identity-protection exchange, and IKE *aggressive mode* refers to the aggressive exchange. They all use Diffie-Hellman Key Agreement technique (DH)[5,6] to generate shared secrets. RFC 2409 defines four different authentication methods for Phase I protocols: preshared key, public key signature, public key encryption, and revised public key encryption. Which method to use is determined by the parameter negotiation. We first discuss the IKE features that are independent of any authentication methods and then the authentication methods.

Common features of IKE. The DH public components generated by the initiator (denoted g^{x_i}) and the responder (denoted g^{x_r}) will be put into the key exchange (KE) payloads.

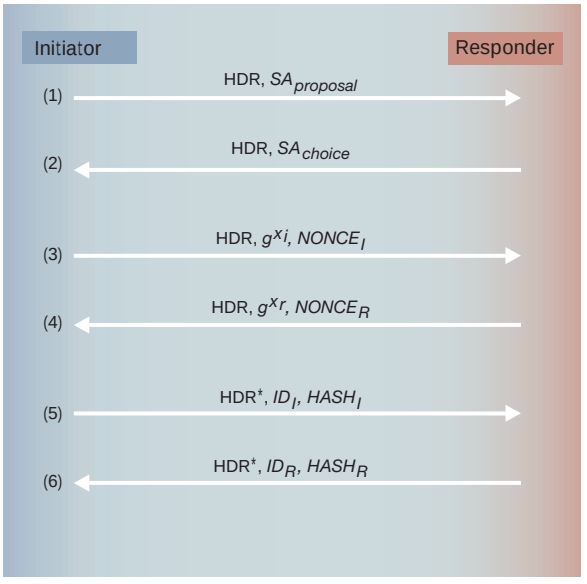
The computation of the authentication data ([*AUTH*] in Figure 1) depends on the particular authentication method used. But regardless of the authentication method, the authentication data are always computed over the following hashes of information:

- $HASH_I = prf(SKEYID, g^{x_i}|g^{x_r}|cookie\text{-}I|cookie\text{-}R|SA|ID_I)$. $HASH_I$ is sent by the initiator.
- $HASH_R = prf(SKEYID, g^{x_r}|g^{x_i}|cookie\text{-}R|cookie\text{-}I|SA|ID_R)$. $HASH_R$ is sent by the responder.

The symbol "|" means concatenation. "SA" in $HASH_I$ and $HASH_R$ is the SA payload sent by the initiator. "Prf" is a pseudorandom function[25] usually implemented by a keyed-hash such as HMAC;[24,36] the exact mathematical transformation is determined during the parameter negotiation. *SKEYID* is the key to prf, it is different for each authentication method. Note that the positions of DH public components and cookies are swapped in $HASH_I$ and $HASH_R$. This gives the two hashes a sense of direction. This idea was first proposed in References 15 and 16 to defeat reflective attacks.

The final output of an IKE Phase I negotiation is an ISAKMP SA with three secret keys shared exclusively between the initiator and the responder. The three keys are:

Figure 7 IKE main mode using preshared key



- $SKEYID_d = prf(SKEYID, g^{x_i x_r}|cookie\text{-}I|cookie\text{-}R|0)$. $SKEYID_d$ is used for deriving other keys in IKE Phases I and II.
- $SKEYID_a = prf(SKEYID, SKEYID_d|g^{x_i x_r}|cookie\text{-}I|cookie\text{-}R|1)$. $SKEYID_a$ is used for authenticating IKE Phase II messages.
- $SKEYID_e = prf(SKEYID, SKEYID_a|g^{x_i x_r}|cookie\text{-}I|cookie\text{-}R|2)$. $SKEYID_e$ is used for encrypting messages 5 and 6 in IKE main mode and all IKE Phase II messages.

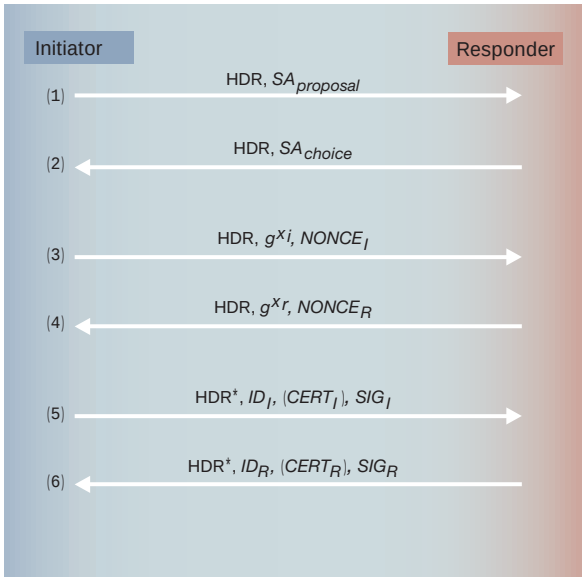
The DH shared secret, $g^{x_i x_r}$, is the main source of entropy (randomness) in deriving these three keys. (Enough entropy is needed to make these keys random and unpredictable.)

IKE defines its own parameters to be used during parameter negotiation. These parameters include authentication methods, hash algorithms, encryption algorithms, pseudorandom functions, and Diffie-Hellman algebraic groups.

We now discuss the four authentication methods. Of these methods, preshared key is the basic form and is discussed first. The other three can be considered variants of the preshared key method.

IKE using preshared key. Figure 7 depicts the IKE main mode using a preshared key. A secret key must be

Figure 8 IKE main mode using public key signature



shared exclusively between the initiator and the responder before the IKE negotiation in Figure 7 takes place. Here [*AUTH*]s are replaced by $HASH_I$ and $HASH_R$. The *SKEYID* is derived from the preshared key:

$$SKEYID = prf(preshared\text{-}key, NONCE_I|NONCE_R)$$

Because of the properties of a pseudorandom function and the fact that the preshared key is an exclusively shared secret, this *SKEYID* is also an exclusively shared secret even though $NONCE_I$ and $NONCE_R$ are sent unencrypted. Since the nonces are fresh, used-only-once random numbers, such an *SKEYID* is also fresh and used only for a particular IKE Phase I negotiation. Therefore, $HASH_I$ and $HASH_R$ provide authentication because the key used to compute them is a fresh, exclusively shared secret. Using *SKEYID* instead of using the preshared key directly also reduces the exposure of the preshared key, which is usually a valuable, long-term secret.

Using the preshared secret key is not scalable, but it does provide the basic operational capability and has no dependency on any PKIX.

IKE using public key signature. Figure 8 depicts the IKE main mode using public key signature.[37] Here [*AUTH*]s are replaced by the initiator's and the responder's signatures, SIG_I and SIG_R, computed over $HASH_I$ and $HASH_R$. $CERT_I$ and $CERT_R$ are the public key certificates of the initiator and the responder. The certificates are placed inside the ISAKMP CERTIFICATE payloads and can be used to verify the signatures. Sending the certificates is optional. If no certificate is sent, then both parties must acquire the other's certificate through some other channel, usually a PKIX. Certificates must be verified before being used to verify signatures. The *SKEYID* is derived from the nonces as follows:

$$SKEYID = prf(NONCE_I|NONCE_R, g^{x_i x_r})$$

The key to the *prf* is $NONCE_I|NONCE_R$, so this *SKEYID* is fresh, but it cannot be used for authentication, because neither the nonces nor the DH secret, $g^{x_i x_r}$, is cryptographically bound to the identity of the initiator or the responder. The authentication is provided by the public key signatures.

Using public key signature eliminates the need for a preshared key and is much more scalable, but it does require at least a minimum PKIX, meaning a certification authority (CA) to issue public key certificates. The CA's own public key certificates can be published and cached everywhere to facilitate verifying other certificates.

IKE using public key encryption. Figure 9 depicts IKE main mode using public key encryption.[38] PK_I and PK_R are the public keys of the initiator and the responder. Note that the nonces are encrypted with the intended receiver's public key. Since only the holder of the corresponding private key can decrypt an encrypted nonce, the nonces become secrets shared between the initiator and the responder. The idea is to use the two nonces to replace a preshared key with the added advantage that the nonces are ephemeral and not long-term shared secrets.

The *SKEYID* is derived from the nonces:

$$SKEYID = prf(hash(NONCE_I|NONCE_R), cookie\text{-}I|cookie\text{-}R)$$

where "*hash*" is a hash algorithm determined during the parameter negotiation. Since the nonces are shared secrets, the *SKEYID* is also a shared secret; therefore, $HASH_I$ and $HASH_R$ can be used directly

for authentication. Also, since no signature or long-term secret is used in authentication, either party can deny the negotiation ever happened, and thus full repudiation[14] of communication is provided. The price for the added advantages is that IKE using public key encryption must have access to an on-line PKIX to get public key certificates. Otherwise, the initiator cannot produce the first key exchange message (message 3 in Figure 9).

The "$hash_1$" term is the hash of the responder's public key certificate containing PK_R. "$hash_1$" is optional and tells the responder which of its public keys is used to encrypt $NONCE_I$. The responder's public key certificate cannot be sent directly because it would reveal the responder's identity. The certificate cannot be encrypted with PK_R because its size is larger than that of PK_R.

ID_I is sent in message 3 instead of 5. This is necessary because the responder needs ID_I to look for the initiator's public key certificate. ID_R is sent in message 4 instead of 6 just to make the flow of information more symmetric.

There are two defects in IKE using public key encryption. First, a total of four expensive public key encryption/decryption operations are used. Since the computation cost of public key encryption and public key signature are of the same order, the computation cost of IKE using public key encryption is about twice that of IKE using public key signature. Second, the responder does not send a hash of PK_I to the initiator. Thus the initiator may have trouble choosing the correct private key to decrypt $\{NONCE_R\}_{PK_I}$. These two defects are corrected in IKE using revised public key encryption.

IKE using revised public key encryption. Figure 10 depicts IKE main mode using revised public key encryption. The *SKEYID* here is the same as that of IKE using public key encryption. IKE using revised public key encryption corrects the two defects of IKE using public key encryption.

First, the total number of public key encryption operations is reduced from four to two. The nonces are still encrypted with public keys, but other encryptions use symmetric keys. The two symmetric encryption keys, Ke_i and Ke_r, are derived as:

$$Ke_i = prf(NONCE_I, cookie\text{-}I)$$

$$Ke_r = prf(NONCE_R, cookie\text{-}R)$$

Figure 9 IKE main mode using public key encryption

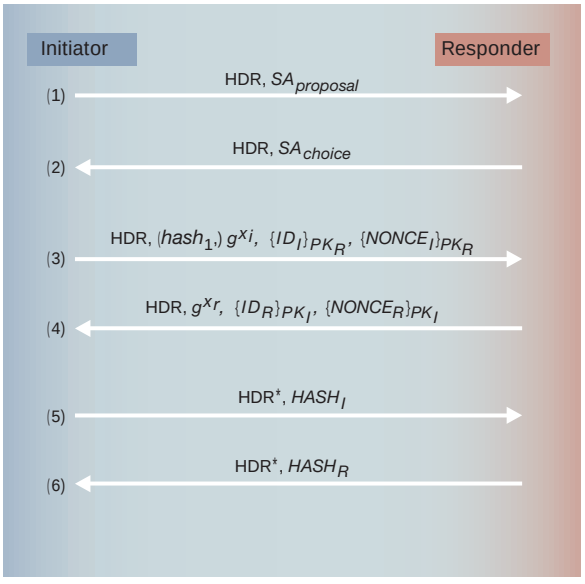
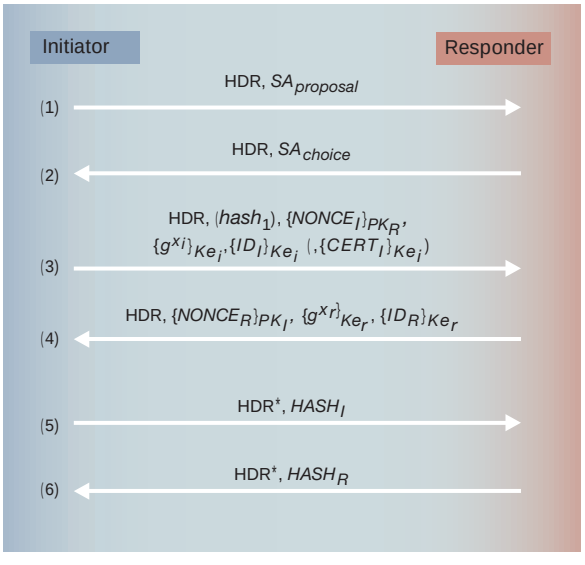


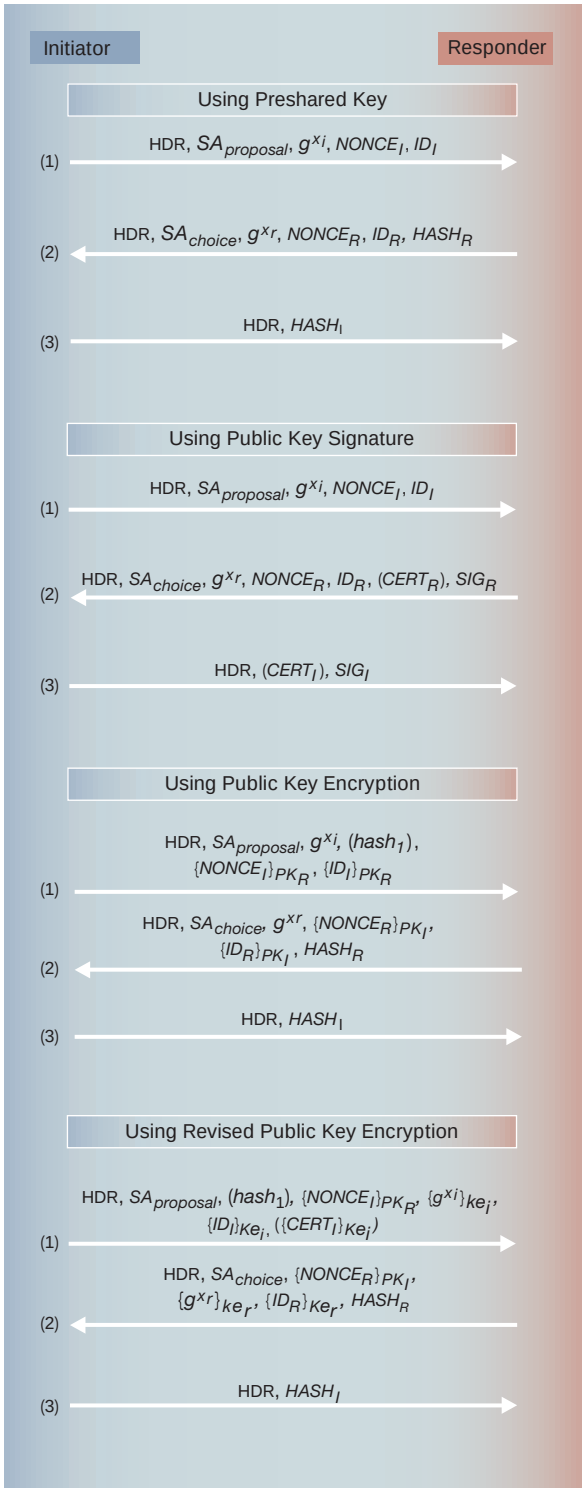
Figure 10 IKE main mode using revised public key encryption



The symmetric key encryption algorithm is determined during the parameter negotiation.

Second, the initiator is allowed to send its encrypted public key certificate to the responder so that the

Figure 11 IKE aggressive mode



responder can use the public key inside the certificate to encrypt $NONCE_R$. The certificate can be encrypted now because symmetric key encryption is used so that the size of the certificate is not an issue.

IKE aggressive mode. Figure 11 depicts IKE aggressive mode using different authentication methods. A few observations can be made: (1) Identities are not encrypted unless (revised) public key encryption is used. (2) The Diffie-Hellman algebraic group cannot be negotiated in IKE aggressive mode. It is chosen by the initiator. (3) The authentication method cannot be negotiated if the initiator chooses to use public key encryption or revised public key encryption. Otherwise, since the forms of the first message are the same when using preshared key or public key signature, the initiator could offer the responder a choice between the two methods.

Security of IKE Phase I protocol. The security of IKE Phase I protocols come from the following two factors:

- Authentication methods—to defeat active, man-in-the-middle, impersonating attacks. Assuming the underlying cryptographic primitives are secure, the security of these authentication methods depends entirely on the security of the preshared secret keys or the ⟨public key, private key⟩ key pairs the methods are using. If these keys are compromised, then any IKE negotiation using the compromised keys is also compromised.

- DH key agreement—to provide the main secret source of entropy, $g^{x_i x_r}$, when deriving the resultant shared secret keys ($SKEYID_d$, $SKEYID_a$, and $SKEYID_e$). Assuming the DH algorithms and the DH algebraic groups used are secure, then the security of DH key agreement depends on keeping x_i and x_r secret. Secrecy requires x_i and x_r to be strong, unpredictable (pseudo)random numbers.

  A secondary secret source of entropy, namely the preshared key or the secret nonces, is provided through *SKEYID* when deriving the shared secrets if IKE uses preshared key, public key encryption, or revised public key encryption. An adversary must compromise both the main source and the secondary source to learn the shared secret keys.

IKE Phase II protocol. IKE defines a new ISAKMP exchange for its Phase II protocol, called *QUICK mode*. Figure 12 depicts IKE QUICK mode. Messages belonging to a particular QUICK mode negotiation are pro-

tected by an ISAKMP SA shared by the initiator and the responder; these messages are identified by two pieces of information in their message headers: ⟨cookie-I, cookie-R⟩ of the ISAKMP SA, and a unique message ID assigned to the negotiation by its initiator.

As the name implies, QUICK mode is meant to be fast and efficient. A QUICK mode negotiation consists of three messages and can be conducted without using any public key cryptography operations. QUICK mode does provide the option of using DH key-agreement techniques, so that g^{x_i} and g^{x_r} in the figure are optional. Also, the identities ID_{ui} and ID_{ur} are optional; they are used by the initiator to indicate a QUICK mode negotiation is a proxy negotiation on behalf of ID_{ui} and ID_{ur}. If the identities are not sent, it is assumed that the unsent identities are the same as ID_I and ID_R in the ISAKMP SA; that is, the two parties are acting on behalf of themselves. Note that if the initiator sends g^{x_i} or (ID_{ui}, ID_{ur}) in the first message, the responder must send g^{x_r} or (ID_{ui}, ID_{ur}) in the second message to continue the negotiation.

In Figure 12, the "*" symbol after the message headers indicates the bodies of these messages are encrypted by the $SKEYID_e$ and its associated encryption algorithm in the ISAKMP SA. $HASH_1$, $HASH_2$, and $HASH_3$ authenticate the corresponding messages. They are computed using $SKEYID_a$ and the pseudorandom function in the ISAKMP SA:

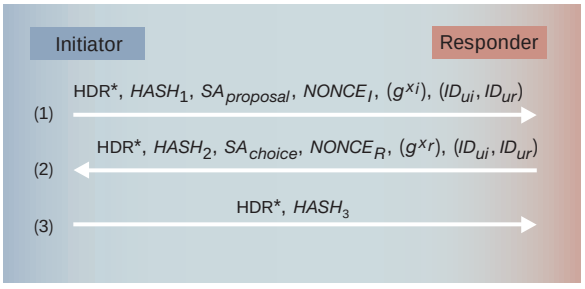
$$HASH_1 = prf(SKEYID_a, message\text{-}ID|SA|NONCE_I[|g^{x_i}][|ID_{ui}|ID_{ur}])$$

$$HASH_2 = prf(SKEYID_a, message\text{-}ID|NONCE_I|SA|NONCE_R[|g^{x_r}][|ID_{ui}|ID_{ur}])$$

$$HASH_3 = prf(SKEYID_a, 0|message\text{-}ID|NONCE_I|NONCE_R)$$

Information in square brackets ([ ]) is optional when computing the hashes; they are used if and only if the messages include them. Since each hash is computed over a different set of information, it has a sense of direction and also a sense of ordering.

Figure 12 IKE Phase II QUICK mode



The proposals in the SA payload are for IPSec SAs. As cited in the earlier subsection on ISAKMP proposal and parameter negotiation, two unidirectional IPSec SAs are generated for each ⟨security protocol, transform⟩ chosen by the responder. The key in such a per-protocol, unidirectional SA, called *KEYMAT*, is derived as:

$$KEYMAT = prf(SKEYID_d, [g^{x_ix_r}|]protocol|SPI|NONCE_I|NONCE_R)$$

"SPI" is the security parameter index. Since each initiator and responder chooses its own SPI for a security protocol, the SPI makes a "KEYMAT" unidirectional. "Protocol" is the identifier for the security protocol and makes a "KEYMAT" per protocol.

By the nature of a pseudorandom function, KEYMAT is not compromised by the compromise of other KEYMATs; thus IKE QUICK mode provides perfect-forward-secrecy in this sense. However, if the protecting ISAKMP SA is compromised, a passive adversary can decipher all QUICK mode negotiations protected by the ISAKMP SA and compromise all KEYMATs generated by these negotiations *unless* DH Key Agreement is used in these negotiations. The use of DH Key Agreement does not prevent an active adversary from impersonating the initiator or the responder when the ISAKMP SA is compromised.

Design and implementation

In this section we discuss the architecture and implementation of our ISAKMP/IKE engine. We first discuss the design rationales and then the architecture itself.

Design rationales. We had to address a number of problems and concerns when we designed and implemented our ISAKMP/IKE engine, including future expansibility, usability and manageability, seemingly infinite number of variations in VPN policies, constantly evolving standards, dependency on public key infrastructure, and code portability. Because of time and other resource constraints, the following decisions were made:

- To focus our effort on implementing the ISAKMP/IKE server executing the ISAKMP/IKE protocol. The implementation of the server should capture the framework concept of ISAKMP to accommodate more than one DOI and therefore the key exchange protocols supported by a DOI. Although only one DOI and one key exchange protocol are currently defined, it is the intention of ISAKMP to accommodate more.
- To isolate the high-level specification of VPN policies from the ISAKMP/IKE server. The high-level specification may take on many different forms, but it eventually has to be translated into the form specified by the ISAKMP/IKE protocol, and this protocol-specific form will be used by the ISAKMP/IKE server in parameter negotiations.
- To isolate public key certificate handling from the ISAKMP/IKE server. Public key infrastructure [12,13] and certificate handling comprise a topic that is very complex by itself, and its standards have been evolving constantly.
- To isolate the run-time management of ISAKMP/IKE from the ISAKMP/IKE server. The ISAKMP/IKE server exports an interface to accept and respond to run-time management commands, such as status query, starting a negotiation, and deleting an SA, but the job of interfacing with the administrators and users to generate run-time management commands and record-keeping is delegated to another process.
- To divide the code into modules based on functionalities to achieve reasonable code stability in the face of constantly evolving protocols. The hope is that a code change resulting from a protocol change will be confined to as few modules as possible, provided that the division of functionalities is correct so that each module exports a stable interface. Object-oriented programming and C++ were used for actual coding; our feeling was that modularity could be achieved more easily in this way.
- To divide the code into a system-independent part and a system-dependent part to provide code portability across different platforms, such as AIX and other UNIX** systems, S/390, and AS/400. The system-independent part should implement the core logic of various functionalities, and the system-dependent part should take care of system chores such as timer and alarm, network communication, signaling, and database storage; it should also export a generic, system-independent application programming interface (API).

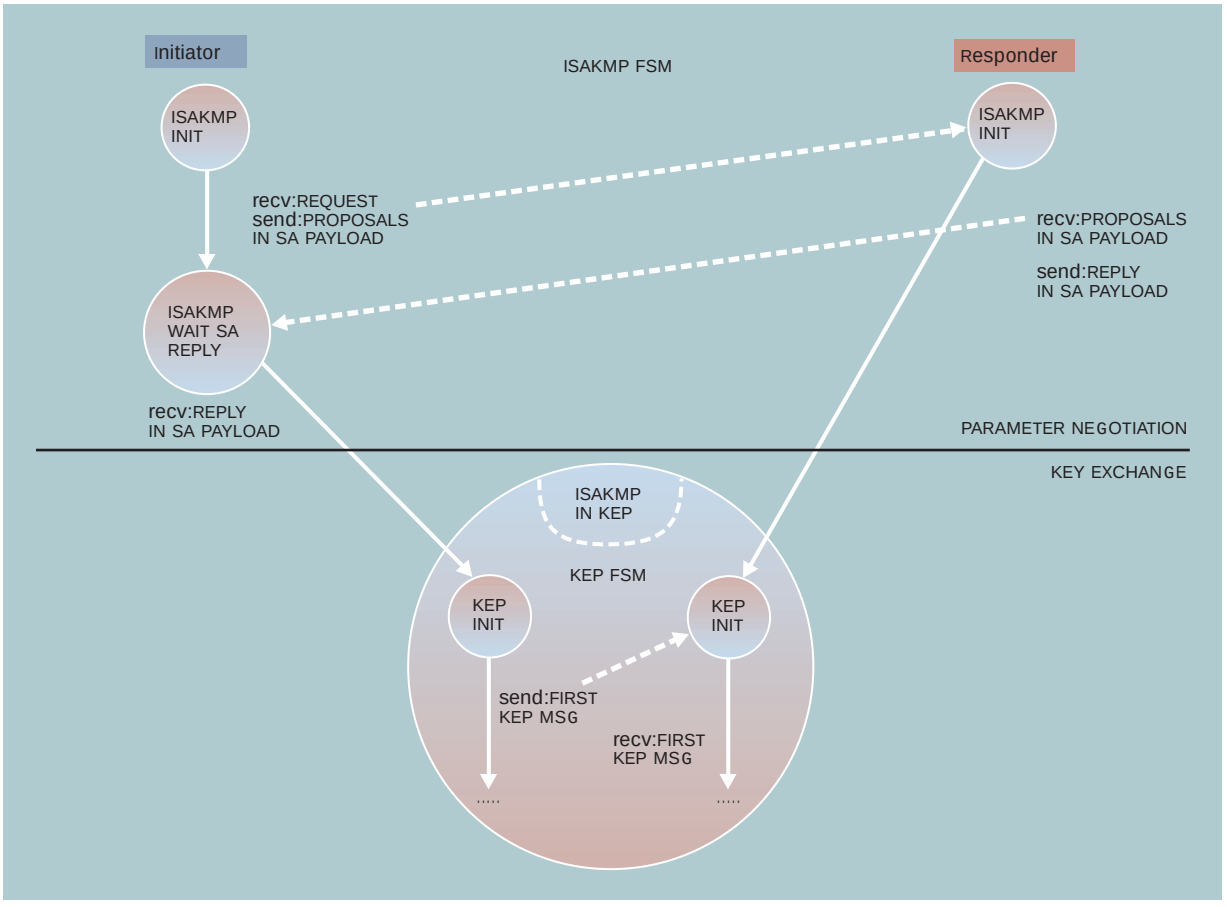
On the basis of the design decisions, the code architecture is divided into four modules: (1) ISAKMP/IKE server: a process implementing the ISAKMP/IKE protocols, (2) Tunnel manager: a front-end process handling administrative commands and user requests, (3) Policy administration tools: a set of utilities to create, store, and maintain VPN policies, and (4) Certificate proxy: a back-end process responsible for acquiring and verifying public key certificates. This division of functionalities provides a lot of freedom for each component to evolve its internal details independently.

We spent most of our effort on the ISAKMP/IKE server, spent some effort on the certificate proxy and the policy administration tools, and did a simple implementation of the tunnel manager. In the following discussion, the amount of details on each module will be given accordingly.

ISAKMP and IKE finite state machines. Communication protocols are usually implemented through finite state machines (FSMs). In the case of ISAKMP/IKE, we think that two FSMs are needed; one implements the ISAKMP framework and the other implements IKE. Figure 13 depicts the combined ISAKMP/IKE FSMs. According to the earlier discussion in the subsection on ISAKMP, ISAKMP FSM is divided into two steps: parameter negotiation and key exchange. Three states are needed to represent the message flow of parameter negotiation, but key exchange is represented in one state. An FSM for a key exchange protocol (KEP), such as the IKE FSM, resides in the ISAKMP key exchange state. This nested FSM design allows FSMs implementing other key exchange protocols to be added to the ISAKMP FSM and therefore preserves the framework concept of ISAKMP. A KEP FSM is defined by the particular KEP and is not constrained by the ISAKMP FSM.

Figure 13 shows parameter negotiation precedes key exchange. If aggressive exchange is used, the two steps will happen in parallel; i.e., the ISAKMP FSM and KEP FSM will exist and operate in parallel.

Figure 13 Nested ISAKMP/KEP finite state machines

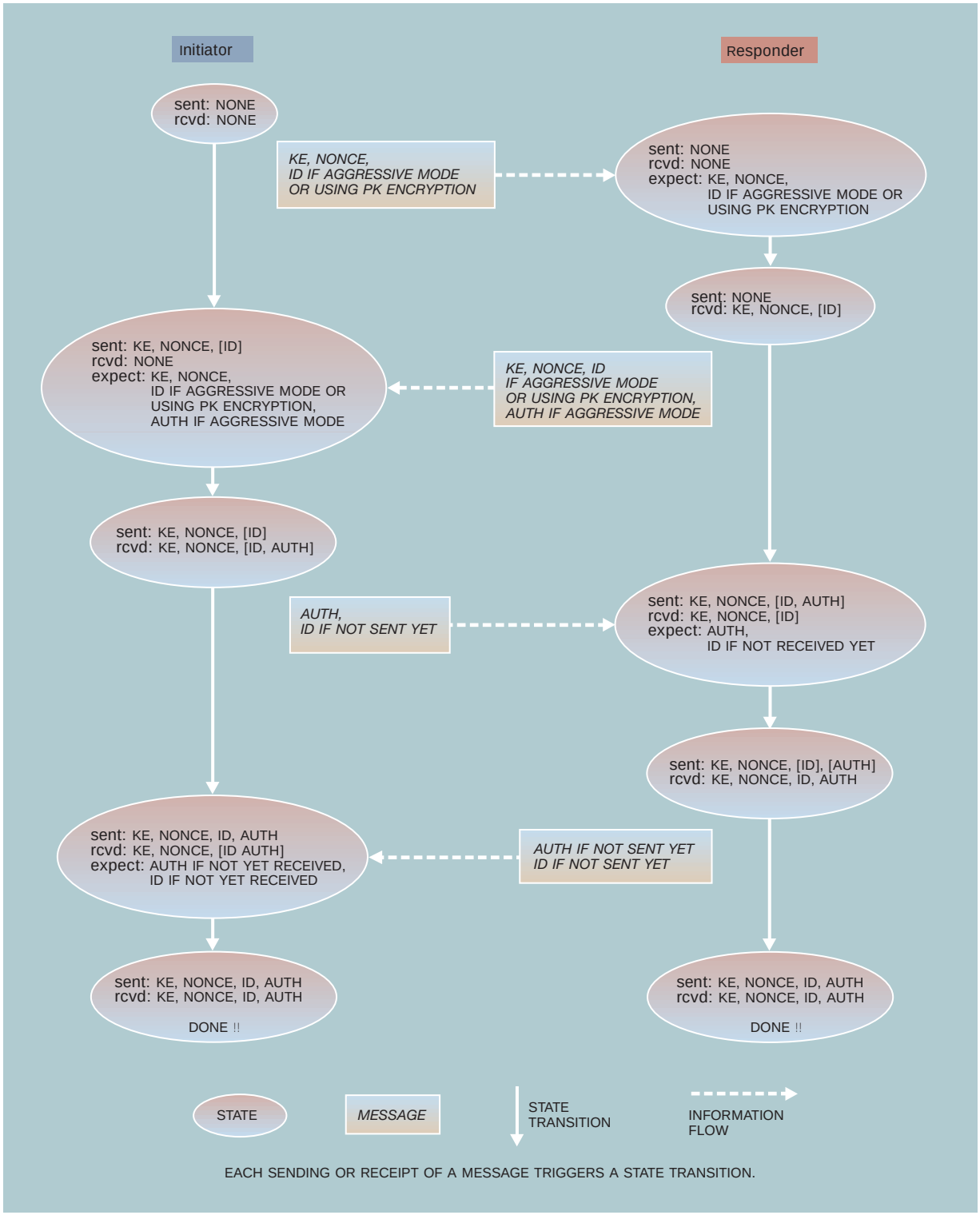


Our ISAKMP FSM is modeled by following the sending and receiving of ISAKMP messages. This approach is not well-suited for modeling the IKE Phase I FSM because IKE Phase I has a main mode and an aggressive mode, and each has its own particular message flow. Also, the types of information carried by a message may differ from one authentication method to another. To avoid implementing an individual FSM for each mode or even for each authentication method, we model the IKE Phase I FSM based on the flow of information rather than on the flow of messages. Regardless of the modes or authentication methods, the initiator and the responder of an IKE Phase I negotiation basically exchange information in the following order:

1. KE and NONCE payloads
2. Identities: ID payload
3. Authentication information: HASH or SIGNATURE (with CERTIFICATE) payloads

In our IKE Phase I FSM, a state is represented by two bit-vectors, the sent vector and the rcvd vector. Each of the KE, NONCE, ID, HASH, and SIGNATURE payloads has a corresponding bit in each vector. The vectors record what payloads have been sent and received. When a message is received, the payloads in the message are examined. Then, on the basis of the two bit-vectors, the role (initiator or responder), the mode, and the authentication method, a decision is made on what actions should be taken: what message to send in response, or that the IKE Phase I negotiation is complete. For an initiator or a responder, an IKE Phase I negotiation is complete if all payloads that should be sent have been sent, all payloads that

Figure 14 IKE Phase I finite state machine



should be received have been received, and the authentication has been verified.

Figure 14 depicts the IKE Phase I FSM. One detail left out in this figure is handling of unexpected messages. If an unexpected message carries a payload of types KE, NONCE, ID, HASH, or SIGNATURE that has already been received, this message is considered a replay. If this payload was carried in the last received message, the response message to that message is resent; otherwise the replay message is simply discarded. A fixed upper limit is set on how many times a message can be resent, after which the IKE Phase I negotiation is simply abandoned to defeat possible denial-of-service attacks. An unexpected message that is not a replay is simply discarded. Note that a replay does not have to be exactly the same as what has been received before. The reason is a message may have been corrupted en route or it may have been manufactured by an attacker. Since there is no way to tell whether a message is genuine until the authentication payloads (HASH or SIGNATURE) are verified, we felt there is no need to do a more time-consuming bit-by-bit comparison. A simple check of the rcvd bit-vector will suffice.

Figure 15 depicts the IKE Phase II FSM. It is designed using the same principle for IKE Phase I FSM, although it is much simpler because it has only one mode and one authentication method.

Code architecture. Figure 16 depicts the run-time architecture of our ISAKMP/IKE code. It is divided into four components as described earlier in this section: tunnel manager, ISAKMP/IKE server, certificate proxy, and policy administration tools.

We focused most of our effort on the ISAKMP/IKE server, so we discuss it first in detail.

ISAKMP/IKE server. The sole purpose of the ISAKMP/IKE server is to conduct ISAKMP/IKE negotiations to generate SAs. The management of a secure tunnel, which is made of a sequence of SAs through the lifetime of the tunnel,[1] is left to the tunnel manager.

This server is implemented in an asynchronous event-driven model. Each event may trigger a state transition in the ISAKMP or IKE FSMs. There are four kinds of events:

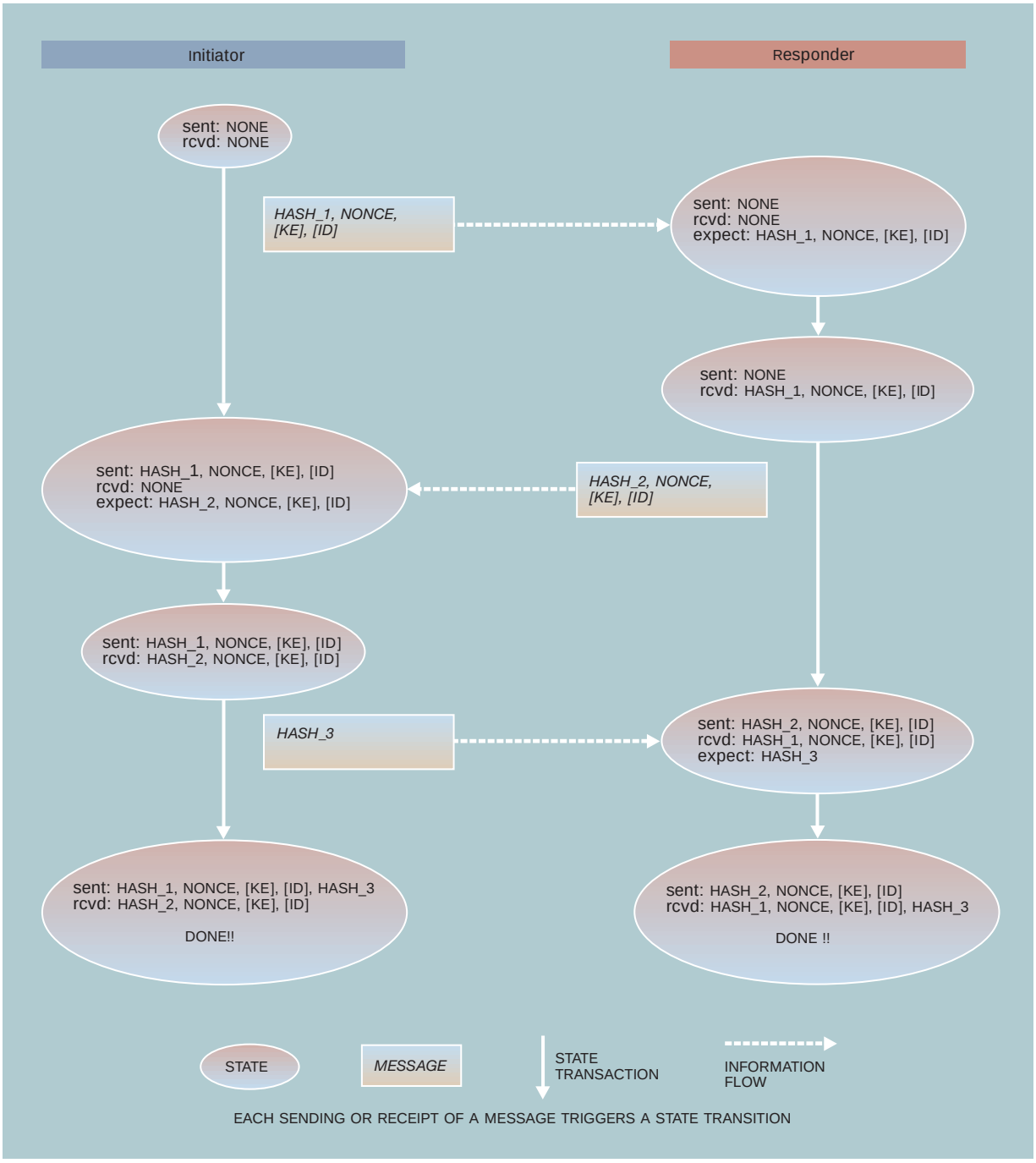
- A request to establish an SA—The response is to start an ISAKMP/IKE negotiation by constructing and sending the first message.
- The receipt of a message—The response is to process the received message and to construct and to send a reply message if necessary. This event may also cause the completion or abandonment of the negotiation.
- The firing of a timer alarm—The response depends on the alarm; usually a message is resent or a negotiation may be abandoned.
- The receipt of a reply from the certificate proxy server—This reply will usually be a verified public key certificate. If the reply indicates an error, the negotiation that made the corresponding request may be terminated.

To achieve portability, we define an object, called an *anchor*, which establishes the system-dependency boundary. Inside the anchor are system-independent objects implementing the logic of the ISAKMP and IKE protocols. Attached externally to the anchor are objects implementing system-independent APIs to system-dependent services.

The system-dependent objects are:

- Network: Sending and receiving messages
- Timer alarm: Setting and firing timer alarms
- Crypto: Providing cryptographic functions, such as DH Key Agreement, encryption, and random number generation. Our implementation uses the RSA BSAFE** library[39] for public key cryptography operations and IBM's code for other functions. The BSAFE library and IBM's code were optimized by our colleagues for the PowerPC* processor architecture.[40] Our implementation uses the C language with a C++ wrapper and is platform-independent. This object may be system-dependent because some platforms may provide hardware-based implementations. Its entropy source for random number generation could also be system-dependent.
- SA cache: Storing Phase II SAs so that the IPSec protocol can use them
- Request capturer: Capturing requests for establishing SAs
- System-dependent parts of DOI as follows:
  - –DOI factory: Generating DOI objects
  - –Per-DOI policy database: Providing policy for parameter negotiation. More discussion on the DOI policy database is given later.
  - –Interface to certificate proxy server
  - –Per-DOI preshared key database: Storing keys preshared with other systems; keys are indexed by the identities or IP addresses of those systems

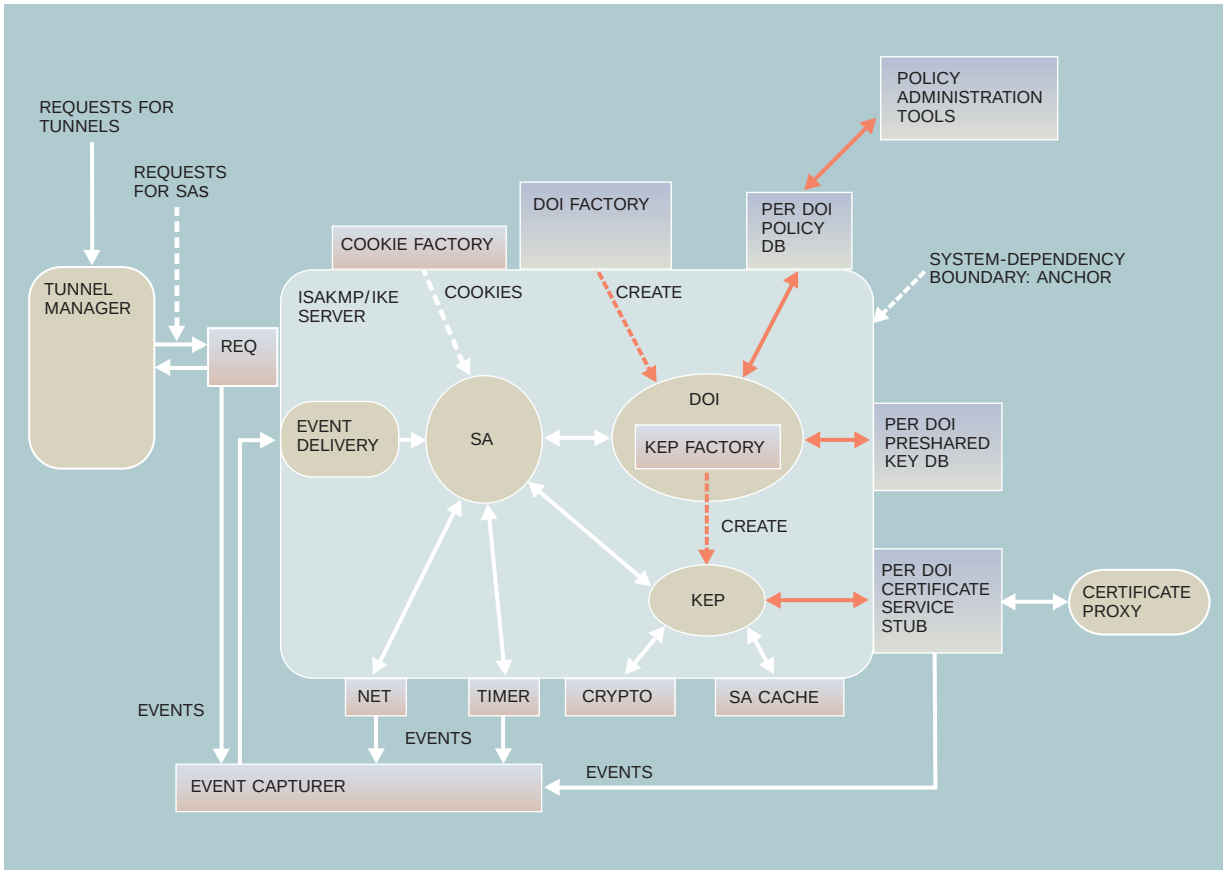
Figure 15 IKE Phase II finite state machine



- Cookie factory: Generating cookie-Is and cookie-Rs. Generation of cookies is system-dependent because the process should include some secrets local to a system.

All events generated by these system-dependent objects are captured by a special event capturer object. The event capturer then delivers captured events to the anchor through the event delivery interface of

Figure 16 ISAKMP/IKE code run-time architecture



the anchor. The event capturer is needed to maintain the system-dependency boundary of the anchor, because events are generated in various system-dependent ways so that the capturer formats these events into a system-independent manner before delivering them to the anchor.

Inside the anchor are three kinds of system-independent objects. One kind is an SA object. An SA object implements an instance of the ISAKMP FSM. One should not confuse an SA object with an SA (security association)—an SA is the result of a negotiation conducted within the context of an SA object. The initiator and the responder of a negotiation each creates an SA object for the negotiation. An SA object creates and invokes a DOI object to conduct parameter negotiation. It creates and invokes a KEP object to conduct key exchange. It receives events from the anchor and processes an event by itself or dispatches the event to either the attached DOI object or the attached KEP object, depending on the state of the ISAKMP FSM. After the negotiation is completed, an SA object exists and holds the final result of the negotiation until the SA object is deleted.

An SA object exports two APIs to conduct negotiation: *start* is invoked by the initiator to construct and send the very first message in a negotiation, *process_msg* processes a received message by itself or by invoking the DOI object or the KEP object.

SA objects are of two types: Phase I and Phase II. A Phase I SA object contains all the Phase II SA objects under its protection.

For an initiator, an SA object is created upon receiving a request to establish an SA; the request includes the responder's identity and the identifier of a DOI. For a responder, an SA object is created upon receiving the first message from the initiator.

The anchor is responsible for dispatching a received message to the proper Phase I SA object, based on the cookies in the message header. A Phase I object is responsible for dispatching a received Phase II message to the proper Phase II SA object, based on the message ID field in the header. A new SA object is created if no proper one exists.

Another system-independent object is a DOI object. A DOI object is mainly used for parameter negotiation. It exports two APIs for parameter negotiation: *propose* is invoked by the initiator to construct a proposal list, and *accept* is invoked by the responder to choose a proposal from the list. An initiator creates a DOI object upon receiving a request to establish an SA. A responder creates a DOI object upon receiving the first message containing the proposal list from the initiator. A link to the per-DOI policy database is built into a DOI object during its creation. A DOI object is always attached to an SA object.

Besides conducting parameter negotiation, a DOI object also provides the following services:

- Storing and operating on the initiator's and the responder's identities. An identity has to be interpreted within the context of a DOI. Parameter negotiation also uses identities to find the correct rules in the policy database.
- Exporting an interface to a per-DOI KEP factory that creates KEP objects. Such a factory exists inside a DOI because the identifier of a KEP has to be interpreted within the context of a DOI.
- Exporting an interface to the per-DOI certificate proxy server to the KEP objects created through the factory
- Exporting an interface to the per-DOI preshared key database

A third system-independent object is a KEP object. A KEP object implements an instance of a KEP FSM. It is always created by an SA object through the per-DOI KEP factory and is attached to the SA object.

A KEP object exports two APIs to conduct key exchange: *Start* constructs the first KEP message sent by the initiator and the first KEP message sent by the responder; *process_msg* processes a received KEP message and usually constructs a reply message. Messages generated by a KEP object are passed to its creating SA object, which will send the message through the network object.

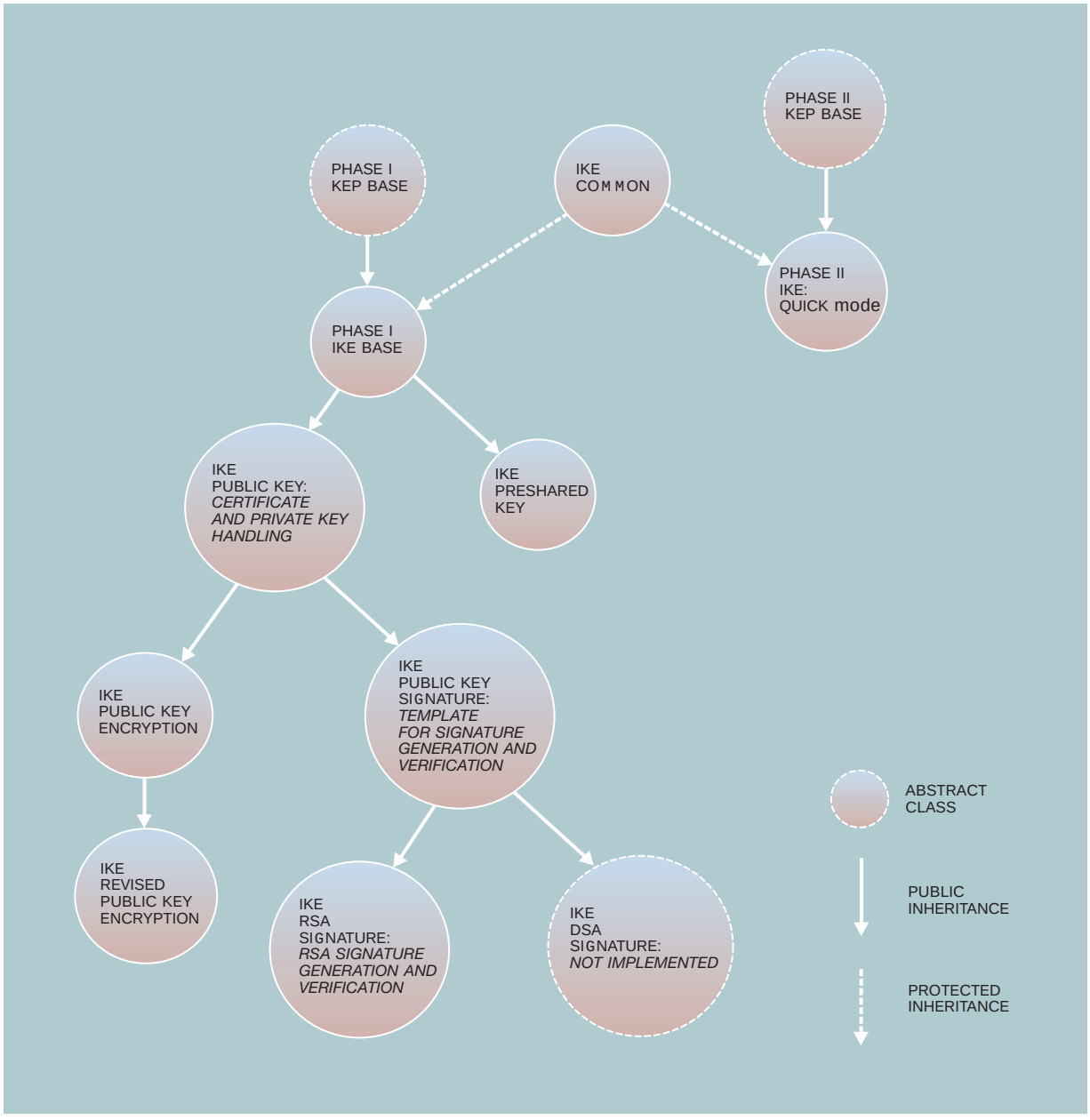
Implementation of IKE FSM. The IKE FSM is implemented as a family of C++ classes. Figure 17 shows the IKE class hierarchy. Phase I and II FSMs are implemented through two separate subhierarchies. Both subhierarchies inherit the *IKE common* class as a C++ protected parent. The *IKE common* class implements functionalities needed by both IKE Phase I and Phase II FSMs, such as nonce generation and storage, Diffie-Hellman Key Agreement, and encryption and decryption of ISAKMP message bodies.

The Phase I hierarchy starts with the *KEP base* abstract class that defines the public interface, including *start* and *process_msg*, of all Phase I KEP classes. The *IKE base* class inherits this interface from the *KEP base* class and implements a template of the IKE Phase I FSM. This template implements the common features of the IKE Phase I protocol, but lets its subclasses override its C++ virtual functions to implement different authentication methods. The *IKE public key* class provides public key certificate handling and private key handling services to its subclasses. The IKE preshared key, IKE public key signature, IKE public key encryption, and IKE revised public key encryption classes implement the four different IKE Phase I authentication methods. Each of the IKE preshared key, IKE public key signature, and IKE public key encryption classes has its own version of a C++ virtual function *compute_SKEYID()* to compute the *SKEYID* of its authentication method; the *IKE revised public key encryption* class inherits *compute_SKEYID()* from the IKE *public key encryption* class. RFC 2409[2] specifies two different public key signature algorithms: RSA (encryption algorithm named for its creators)[41,42] and DSA (Digital Signature Algorithm).[43,44] We only implemented the RSA algorithm through the *IKE RSA signature* class.

Figures 18 and 19 show the flowcharts of the *start()* and the *process_msg()* functions of the *IKE base* class.

The *start()* function is invoked by the initiator and the responder of a negotiation to build the first pair of IKE messages exchanged between them. The *authn_specific_start()* function is a C++ virtual function that can be overridden by subclasses to do processing specific to a particular authentication method. It was originally envisioned to be used for the public key encryption methods to encrypt individual payloads. We later discovered that writing new *start()* functions for the *IKE public key encryption* and *IKE revised public key encryption* classes is better for performance and results in simpler code. However,

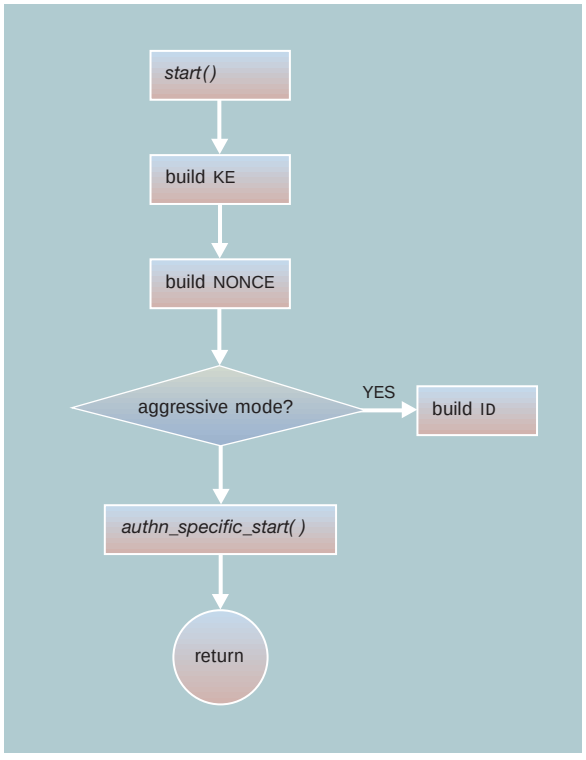
Figure 17 IKE class hierarchy



the concept for processing specific to an authentication method is still kept.

The *process_msg( )* function is used to process received IKE messages. Whereas *start( )* begins an IKE Phase I FSM, *process_msg( )* keeps the FSM going by processing received messages, generating reply messages, and finally concluding the negotiation. Preauthentication processing in Figure 19 refers to any computation that is needed to generate or verify the authentication data, including computing *SKEYID*, and in the case of IKE main mode, com-

Figure 18 IKE Phase I *start()* flowchart



puting $SKEYID_d$, $SKEYID_a$, and $SKEYID_e$. The following four C++ virtual functions, whose implementations are authentication-method-specific, are shown in Figure 19:

- *authn_specific_process( )*: It is used to decrypt payloads if (revised) public key encryption authentication methods are used. It is also used to issue a prefetching of public key certificates if an authentication method requires such certificates. It may be used for other processing if new authentication methods are added to IKE.
- *needRemoteCred( )*: It is used to determine whether a credential of the remote party is needed at the current state of the IKE FSM. For the authentication methods discussed earlier, a credential may be a preshared key or a public key certificate.
- *getRemoteCred( )*: It is used to obtain the remote party's credential. It may either access the local preshared key database or ask the certificate proxy to obtain a public key certificate.
- *verify_auth( )*: It is used to verify the authentication data. The data are carried in either a HASH payload or a SIGNATURE payload. Not shown in Figure 19 is the function *compute_auth( )* that computes the authentication data and puts the data in the proper payload. For each implementation of *verify_auth( )* there is a corresponding implementation of *compute_auth( )*. *Compute_auth( )* is invoked by the *I_reply( )* and *R_reply( )* functions that build reply messages for the initiator and the responder, respectively.

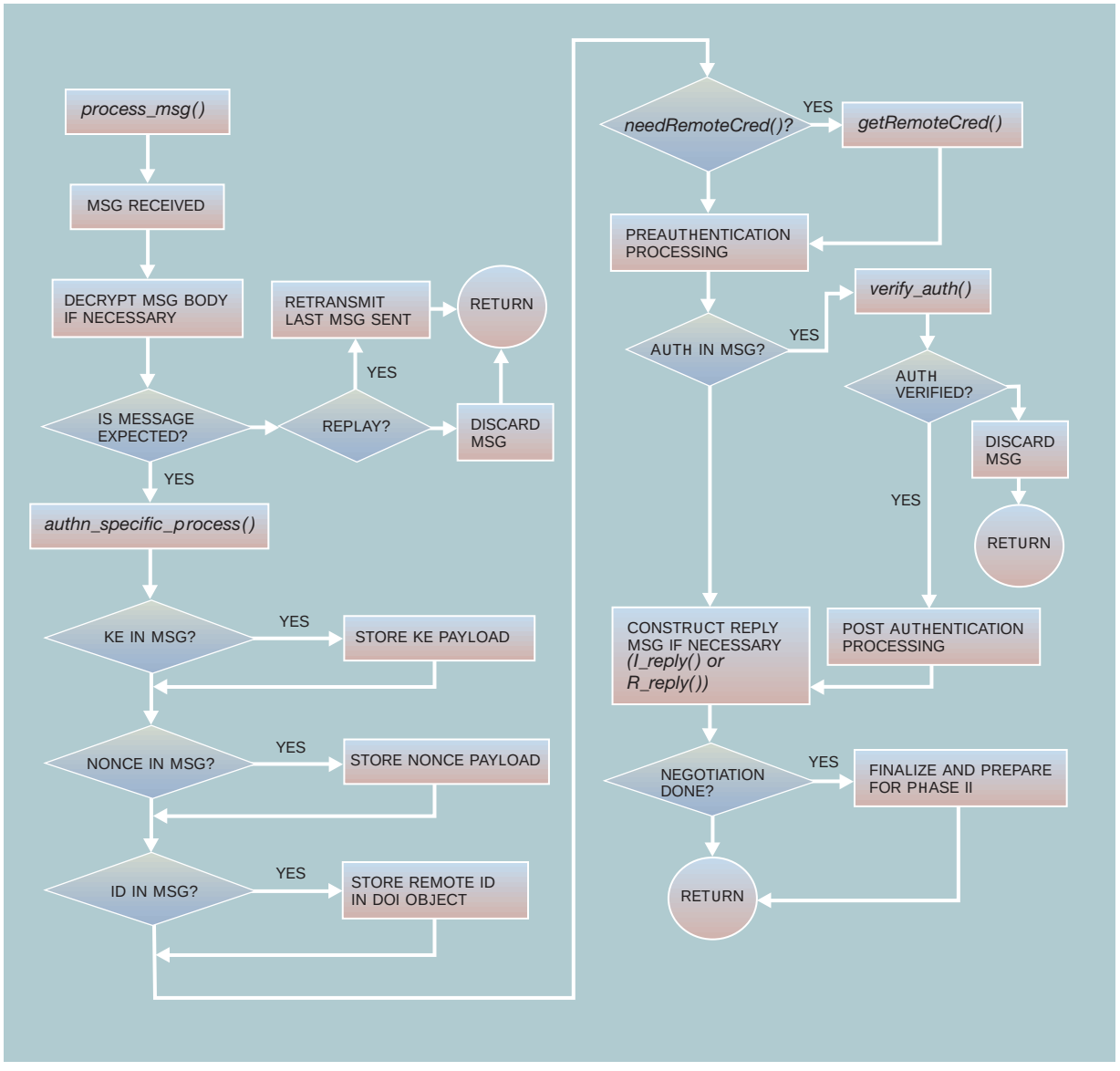
Implementation of ISAKMP/IKE policy database. Figure 20 shows the schema for the ISAKMP/IKE policy database.

In the schema, a remote identifier (ID) references a pair of lists of proposals. Each list of proposals contains references to proposals. List for Initiator is used by an initiator to construct an ISAKMP proposal list to the responder identified by the remote ID; the construction process basically copies the list. List for Responder is used by a responder to match against a proposal list sent by the initiator identified by the remote ID. The matching process takes a proposal from the initiator's proposal list, starting from the first, and matches the proposal against each proposal in the List for Responder, starting from the first. The matching process continues until the first match is found, or until the initiator's proposal list is exhausted and no match is found, in which case the responder will send a "no match found" error message to the responder. Otherwise, the initiator's matching proposal will be sent back as the reply.

If IKE main mode (ISAKMP identity-protection exchange) is used, then the initiator's ID is not available to the responder when the matching takes place, and the initiator's IP address is used instead.

It is open to debate as to whether two different proposal lists for a remote ID are really needed. Our argument is that during a Phase I parameter negotiation the responder may know much less about an initiator than the initiator knows about the responder, as in the IKE main mode case. Therefore, the responder may need to be more cautious and use a different list. This argument is not very strong, but neither can we provide any strong argument that the two-list schema will never be needed; so we stick with this schema. If only one list is needed, then the two references in a pair can reference the same list. We also feel that the ultimate goal of IKE is to generate Phase II SAs to protect data communication, so it is acceptable to be overly cautious in Phase I. A Phase II negotiation can always decide on a more

Figure 19 IKE Phase I *process_msg ( )* flowchart



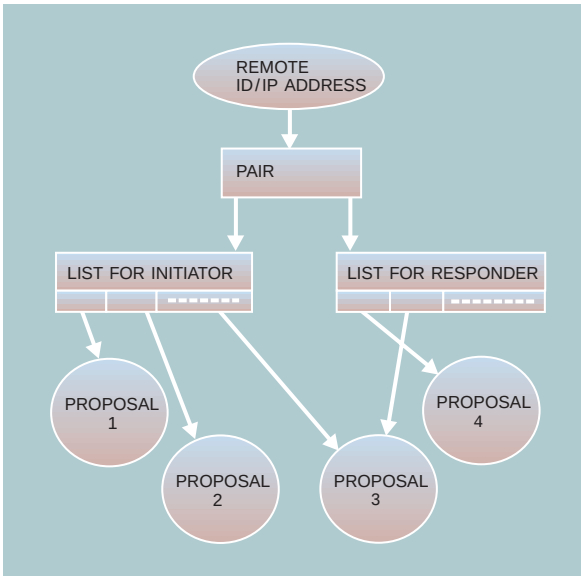
suitable proposal because IDs of both sides are readily available.

Certificate proxy. The certificate proxy process provides the following three services to the ISAKMP/IKE server:

- Acquire and verify a public key certificate specified by certificate type, usage, owner's (subject) identity, etc.
- Verify a given public key certificate
- Verify a given public key certificate and that the certificate belongs to a specific identity

Our implementation of the certificate proxy is not simple because of the complexity of certificate handling. It is rather experimental because the standards for certificate handling and for IPSec/IKE certificates were still evolving when the code was being written. IBM's product divisions have since greatly enhanced

Figure 20 ISAKMP/IKE policy schema



the certificate proxy and brought it more in line with the standard. The standards are still not finalized.

Policy administration tools. The policy administration tools are parsers that parse four kinds of input files that are an instance of the policy database described earlier. They are: remote-ID to pair mapping, pairs of references to lists of references to proposals, lists of references to proposals, and proposals.

The input files are all stanza files in plain text; they are all created by standard text editors. The output files are simple database tables in the popular UNIX NDBM format. The output files are used by the DOI objects for parameter negotiations.

IBM's product divisions have enhanced these tools and the policy database by adding a graphical user interface and migrating the database to DB2* (DATABASE 2*) to provide much better manageability and usability.

Tunnel manager. Our tunnel manager is a simple command-line utility that constructs a request to establish an SA and sends the request to the ISAKMP/IKE server.

IBM's product divisions have greatly enhanced the tunnel manager so it becomes the central control point for managing secure tunnels. The enhanced tunnel manager can determine whether an existing secure tunnel can satisfy a request. It can also provide status information on all existing tunnels and ongoing negotiations.

Performance

Our colleagues from the AIX VPN team have done performance measurements on a 350MHz PowerPC-based[40] system. The measurements were made with the following parameters:

- Phase I:
  - –DH Key Agreement: IKE DH group 1 (768-bit prime modulus)
  - –Encryption: DES
  - –Hash and pseudorandom function: MD5 and HMAC-MD5
  - –Mode: aggressive
  - –Authentication method: preshared key
- Phase II: no PFS (DH). Note that a Phase II negotiation applies the algorithms of its Phase I negotiation for encryption, hash, and pseudorandom functions.

The ISAKMP/IKE code was instrumented to generate time stamps with microsecond resolution. The policy databases of the initiator and the responder contain only one list with one proposal defined according to the configuration parameters. Ten Phase I negotiations were performed. For each Phase I SA generated, two Phase II negotiations were performed under its protection.

The measurements show that a Phase I negotiation takes about 40035 microseconds and a Phase II negotiation takes about 3740 microseconds. Further investigation shows that a modular exponentiation in IKE DH group 1 takes 19000 microseconds. Since the responder or the initiator needs to perform two modular exponentiations to complete a DH Key Agreement, it accounts for 95 percent of the time (38000 microseconds) spent in a Phase I negotiation.

Acknowledgment

The author would like to thank Hugo M. Krawczyk for his constant encouragement and guidance, without which the work could not have been done; he and Ran Canetti also made contributions to the IKE protocol specification and provided the author with very useful advice on cryptography. Dimitrios Pendarakis implemented IKE Phase I using public key

encryption and revised public key encryption and IKE QUICK mode. Pankaj Rohatgi provided the very fast DES implementation for the AIX PowerPC platform. Suresh N. Chari provided the optimized PowerPC code for modular exponentiation in the AIX version of the BSAFE library and provided its performance figures. Padmaja Rao and Ravi Narayan provided the core code for handling public key certificates. The author would also like to thank Hamid Ahmadi, Matthias Kaiserswerth, Arvind Krishna, Charles C. Palmer, David R. Safford, and Josyula R. Rao for their management support for this work and to thank IBM colleagues in the IBM product divisions for their cooperation and efforts.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of X/Open Company Limited or RSA Security, Inc.

Cited references and notes

1. P.-C. Cheng, J. A. Garay, A. Herzberg, and H. Krawczyk, "A Security Architecture for the Internet Protocol," *IBM Systems Journal* **37**, No. 1, 42–60 (1998).
2. D. Harkins and D. Carrel, *The Internet Key Exchange (IKE)*, The Internet Society, RFC 2409 (November 1998).
3. IETF IP Security Protocol Working Group, http://www.ietf.org/html.charters/ipsec-charter.html.
4. S. Kent and R. Atkinson, *Security Architecture for the Internet Protocol*, The Internet Society, RFC 2401 (November 1998).
5. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York (1996).
6. W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory* **IT-22**, No. 6, 644–654 (November 1976).
7. D. Maughan, M. Schneider, M. Schertler, and J. Turner, *Internet Security Association and Key Management Protocol (ISAKMP)*, The Internet Society, RFC 2408 (November 1998).
8. D. Piper, *The Internet IP Security Domain of Interpretation for ISAKMP*, The Internet Society, RFC 2407 (November 1998).
9. J. Postel, *User Datagram Protocol*, The Internet Society, RFC 768 (August 1980).
10. W. Diffie, P. van Oorschot, and M. Wiener, "Authentication and Authenticated Key Exchanges," *Designs, Codes and Cryptography* **2**, 107–125 (1992).
11. B. Schneier, *Applied Cryptography*, *2nd Edition*, John Wiley & Sons, Inc., New York (1996).
12. R. Housley, W. Ford, T. Polk, and D. Solo, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, The Internet Society, RFC 2459 (January 1999).
13. IETF Public-Key Infrastructure (X.509) Working Group. http://www.ietf.org/html.charters/pkix-charter.html.
14. H. Krawczyk, "SKEME: A Versatile Secure Key Exchange Mechanism for Internet," *The Proceedings of the 1996 Internet Society Symposium on Network and Distributed Systems Security* (February 1996), pp. 114–127.
15. R. Bird, I. Gopal, A. Herzberg, P. A. Janson, S. Kutten, R. Molva, and M. Yung, "The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution," *IEEE/ACM Transactions on Networking* **3**, No. 1, 31–41 (February 1995).
16. R. Bird, I. Gopal, A. Herzberg, P. A. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic Design of a Family of Attack-Resistant Authentication Protocols," *IEEE Journal on Selected Areas in Communications* **11**, No. 5, 679–693 (June 1993).
17. P. Cheng, J. A. Garay, A. Herzberg, and H. Krawczyk, *Modular Key Management Protocol*, IETF ⟨draft-cheng-modular-ikmp-00.txt⟩ (November 1994).
18. P.-C. Cheng, J. A. Garay, A. Herzberg, and H. Krawczyk, "Design and Implementation of Modular Key Management Protocol and IP Secure Tunnel on AIX," *The Proceedings of the 5th USENIX UNIX Security Symposium* (June 1995), pp. 41–54.
19. S. Kent and R. Atkinson, *IP Encapsulating Security Payload (ESP)*, The Internet Society, RFC 2406 (November 1998).
20. S. Kent and R. Atkinson, *IP Authentication Header*, The Internet Society, RFC 2402 (November 1998).
21. C. Madson and R. Glenn, *The Use of HMAC-SHA-1-96 Within ESP and AH*, IETF, RFC 2404 (November 1998).
22. C. Madson and R. Glenn, *The Use of HMAC-MD5-96 Within ESP and AH*, IETF, RFC 2403 (November 1998).
23. P. Karn and W. A. Simpson, *The Photuris Session Key Management Protocol*, IETF ⟨draft-ipsec-photuris-02.txt⟩ (July 1995).
24. H. Krawczyk, M. Bellare, and R. Canetti, "Keyed Hash Functions and Message Authentication," *Proceedings of Crypto'96*, http://www.research.ibm.com/security/keyed-md5.html (1996).
25. M. Bellare, R. Canetti, and H. Krawczyk, "Pseudorandom Functions Revisited: The Cascade Construction and Its Concrete Security," *37th Annual Symposium on the Foundations of Computer Science*, IEEE (1996).
26. *Internet Protocol*, The Internet Society, RFC 791 (September 1981).
27. P. Mockapetris, *Domain Names—Concepts and Facilities*, The Internet Society, RFC 1034 (November 1987).
28. T. T. Pummill and B. Manning, *Variable Length Subnet Table for IPv4*, The Internet Society, RFC 1878 (December 1995).
29. S. E. Deering and R. M. Hinden, *Internet Protocol*, *Version 6 (IPv6) Specification*, The Internet Society, RFC 2460 (December 1998).
30. *Data Communication Networks Directory*, International Telecommunication Union Recommendations X.500–X.521 (1989).
31. T. Berners-Lee, R. T. Fielding, and L. Masinter, *Uniform Resource Identifiers (URI): Generic Syntax*, The Internet Society, RFC 2396 (August 1998).
32. *Transmission Control Protocol*, The Internet Society, RFC 793 (September 1981).
33. J. Postel and J. Reynolds, *TELNET Protocol Specification*, The Internet Society, RFC 854 (May 1983).
34. *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, U.S. Department of Defense (August 1983).
35. H. K. Orman, *The OAKLEY Key Determination Protocol*, The Internet Society, RFC 2412 (November 1998).
36. H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, The Internet Society, RFC 2104 (February 1997).
37. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *op. cit*., Chapter 11.
38. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *op. cit*., Chapter 8.