Efficient, DoS-Resistant, Secure Key Exchange for Internet Protocols

William Aiello[1], Steven M. Bellovin[1], Matt Blaze[1], Ran Canetti[2], John Ioannidis[1], Angelos D. Keromytis[3], and Omer Reingold[1]

[1] AT&T Labs – Research, {*aiello,smb,mab,ji,omer*}*@research.att.com*
[2] IBM T.J. Watson Research Center, *canetti@watson.ibm.com*
[3] Columbia University in the City of New York, *angelos@cs.columbia.edu*

1 Introduction

Many public-key-based key setup and key agreement protocols already exist and have been implemented for a variety of applications and environments. Several have been proposed for the IPsec protocol, and one, IKE[1], is the current standard. IKE has a number of deficiencies, the three most important being that the number of rounds is high, that it is vulnerable to denial-of-service attacks, and the complexity of its specification. (This complexity has led to interoperability problems, so much so that, several years after its initial adoption by the IETF, there are still completely non-interoperating implementations.)

While it may be possible to "patch" the protocol to fix some of these problems, we would prefer to replace IKE with something better. With that in mind, we set out to engineer a new key exchange protocol specifically for Internet security applications. With a view toward its possible role as a successor to IKE, we call our new protocol "JFK," which stands for "Just Fast Keying."

1.1 Design Goals

We seek a protocol with the following characteristics:

Security: No one other than the participants may have access to the generated key.
Simplicity: It must be as simple as possible.
Memory-DoS: It must resist memory exhaustion attacks on the responder.
Computation-DoS: It must resist CPU exhaustion attacks on the responder.
Privacy: It must preserve the privacy of the initiator.
Efficiency: It must be efficient with respect to computation, bandwidth, and number of rounds.
Non-Negotiated: It must avoid complex negotiations over capabilities.
PFS: It must approach perfect forward secrecy.

The *Security* property is obvious enough; the rest, however, require some discussion.

The *Simplicity* property is motivated by several factors. Efficiency is one; increased likelihood of correctness is another. But our motivation is especially colored by our experience with IKE[1]. Even if the protocol is defined correctly, it must be implemented correctly; if a protocol definition is too complex, implementors will get it wrong. This hinders both security and interoperability.

The *Memory-DoS* and *Computation-DoS* properties have become more important in the context of recent Internet denial-of-service attacks. Photuris [2] was the first published key management protocol for which DoS-resistance was a design consideration; we suggest that these properties are at least as important today.

The *Privacy* property means that the protocol must not reveal the initiator's identity to any unauthorized party, including an active attacker that attempts to act as the responder. Protecting the responder's privacy does not appear to be of much value, except perhaps in peer-to-peer communication: in many cases, the responder is a server with a fixed address or characteristics (*e.g.,* well-known web server). A third approach is to allow for a protocol that allows the two parties to negotiate who needs identity protection. In JFK, we decided against this approach: it is unclear what, if any, metric can be used to determine which party should receive identity protection; furthermore, this negotiation could act as a loophole to make initiators reveal their identity first.

The *Efficiency* property is worth discussing. In many protocols, key setup is must be performed frequently enough that it can become a bottleneck to communication. The key exchange protocol must minimize both computation as well total bandwidth and round trips. Round trips can be an especially important factor over unreliable media.

The *Non-Negotiated* property is necessary for several reasons. The first, of course, is as a corollary to *Simplicty* and *Efficiency.* Negotiations create complexity and round trips, and hence should be avoided. Denial of service resistance is also relevant here; a partially-negotiated security association is consuming resources.

The *PFS* property is perhaps the most controversial. Rather than assert that "we must have perfect forward secrecy at all costs", we treat the *amount* of forward secrecy as an engineering parameter that can be traded off against other necessary functions, such as resistance to denial-of-service attacks. In fact, this corresponds quite nicely to the reality of today's Internet systems, where a compromise during the existence of a security association will reveal the plaintext of any ongoing transmissions. Our scheme has the concept of a "forward secrecy interval"; associations are protected against compromises that occur outside of that interval.

Protocol design is, to some extent, an engineering activity, and we need to provide for trade-offs between different types of security. There are trade-offs that we made during the protocol design, and others, such as the trade-off between forward secrecy and computational effort, that are left to the implementation and to the user, *e.g.,* selected as parameters during configuration and session negotiation.

2 Protocol Definition

2.1 Notation

First, some notation:

$\{M\}_k$	Encryption of message M with symmetric key k.
$H_k(M)$	Keyed hash (e.g., HMAC [3]) of message M using key k.
$S_x[M]$	Digital signature of message M with the private key belonging to principal x (Initiator or Responder). It is assumed to be a non-message-recovering signature.

The message components used in JFK are:

g^x	Diffie-Hellman exponentials; also identifying the group-ID.
g^i	Initiator's current exponential.
g^r	Responder's current exponential.
N_I	Initiator nonce, a random bit-string.
N_R	Responder nonce, a random bit-string.
sa	defines cryptographic and service properties of the security association (SA) that the Initiator wants to establish. It contains a Domain-of-Interpretation which JFK understands, and an application-specific bitstring.
sa$'$	SA information the Responder may need to give to the Initiator (*e.g.,* the Responder's SPI in IPsec).
HK$_r$	A transient hash key private to the Responder.
K_{ir}	shared key derived from g^{ir}, N_I, and N_R used for protecting the application (*e.g.,* IPsec SA).
K_e	shared key derived from g^{ir}, N_I, and N_R used to protect messages 3 and 4 of the protocol.
ID$_I$	Initiator's certificates or public-key identifying information.
ID$_R$	Responder's certificates or public-key identifying information.
grpinfo$_R$	all groups supported by the Responder, the symmetric algorithm used to protect messaged 3 and 4, and the hash function used for key generation.

Both parties must pick a fresh nonce at each invocation of the JFK protocol. The nonces are used in the session key computation in order to provide key independence when one or both prties reuse their Diffie-Hellman exponential; the session key will be different different between independent runs of the protocol, as long as one of the nonces or exponentials changes.

HK$_R$ is a global parameter for the Responder – it stays the same between protocol runs, but it can change periodically. The Responder must bick a new g^r every time HK$_R$ changes.

2.2 The Protocol

The basic JFK protocol consists of four messages, for a total of two round trips.

$$I \rightarrow R : N_I, g^i \pmod p \tag{1}$$

$$R \rightarrow I : N_I, N_R, g^r, \mathrm{grpinfo}_R, \mathrm{ID}_R, S_R[g^r], H_{\mathrm{HK}_R}(N_I, N_R, g^i, g^r) \tag{2}$$

$$I \rightarrow R : N_I, N_R, g^i, g^r, H_{\mathrm{HK}_R}(N_I, N_R, g^i, g^r), \{\mathrm{ID}_I, \mathrm{sa}, S_I[N_I, N_R, g^i, g^r, \mathrm{ID}_R, \mathrm{sa}]\}_{K_e} \tag{3}$$

$$R \rightarrow I : \{S_R[N_I, N_R, g^i, g^r, \mathrm{ID}_I, \mathrm{sa}, \mathrm{sa}'], \mathrm{sa}'\}_{K_e} \tag{4}$$

The key used to protect Messages (3) and (4), K_e, is computed as $H_{g^{ir}}(N_I, N_R, 1)$. The session key used by IPsec (or any other application), K_{ir}, is $H_{g^{ir}}(N_I, N_R, 0)$.

Message (1) is straightforward; note that it assumes that the Initiator already knows a group and generator that is acceptable to the Responder. The Initiator can reuse a g^i value in multiple instances of the protocol with the Responder, or other responders that accept the same group, for as long as she wishes her forward secrecy interval to be. We discuss how the Initiator can discover what groups to use later.

Message (2) is more complex. Assuming that the Responder accepts the Diffie-Hellman group in the Initiator's message (rejections are discussed in Section 2.4), he replies with a signed copy of his own exponential (in the same group, also $\pmod p$), information on what secret key algorithms are acceptable for the next message, a random nonce, his identity (certificates or a string identifying his public key), and an authenticator calculated from a secret, HK_R, known to the Responder; the authenticator is computed over the two exponentials and nonces. The authenticator key is changed at least as often as g^r, thus preventing replays of stale data. The Responder's exponential may also be reused; again, it is regenerated according to the Responder's forward secrecy interval. The signature on the exponential needs to be calculated at the same rate as the Responder's forward secrecy interval (when the exponential itself changes). Finally, note that the Responder does not need to generate any state at this point, and the only expensive operation is a MAC calculation.

Message (3) echoes back the data sent by the Responder, including the authenticator. The authenticator is used by the Responder to verify the authenticity of the returned data. The message also includes the Initiator's identity and service request, and a signature computed over the nonces, the Responder's identity, and the two exponentials. This latter information is all encrypted under a key derived from the Diffie-Hellman computation and the nonces N_I and N_R. The encryption and authentication use algorithms specified in $\mathrm{grpinfo}_R$. The Responder keeps a copy of recently-received Message (3)s, and their corresponding Message (4). Receiving a duplicate (or replayed) Message (3) causes the Responder to simply retransmit the corresponding Message (4), without creating new state or invoking IPsec. This cache of messages can be reset as soon as g^r or HK_R are changed. The Responder's exponential (g^r) is re-sent by the Initiator because the Responder may be generating a new g^r for every new JFK protocol run (*e.g.,* if the arrival rate of requests is below some threshold). It is important that the responder deal with repeated Message (3)s as described above. Responders that create new state for a repeated Message (3) open the door to attacks against the protocol.

Note that the signature is protected by the encryption. This is necessary, since everything signed is public except the sa, and that is often guessable. An attacker could verify guesses at identities if the signature were not encrypted.

Message (4) contains application-specific information (such as the Responder's IPsec SPI), and a signature on both nonces, both exponentials, and the Initiator's identity. Everything is encrypted by a K_e, which is derived from N_I, N_R, and g^{ir}. The encryption algorithm is specified in $\mathrm{grpinfo}_R$.

2.3 Discussion

The design follows from our requirements. With respect to communication efficiency, observe that the protocol requires only two round trips. The protocol is optimized to protect the Responder against denial of service attacks on state or computation. The Initiator bears the initial computational burden and must establish round-trip communication with the Responder before the latter is required to perform expensive operations. At the same time, the protocol is designed to limit the private information revealed by the Initiator; she does not reveal her identity until she is sure that only the Responder can retrieve it. (An active attacker can replay an old Message (2) as a response to the Initiator's initial message, but he cannot retrieve the Initiator's identity from Message (3) because he cannot complete the Diffie-Hellman computation).

The Initiator's initial message, Message (1), is a straightforward Diffie-Hellman exponential. Note that this is assumed to be encoded in a self-identifying manner, *i.e.,* it contains a tag indicating which modulus and base was used. The nonce N_I serves two purposes: first, to allow the Initiator to reuse the same exponential across different sessions (with the same or different Responders, within the Initiator's forward secrecy interval) while ensuring that the resulting session key will be different. Secondly, it can be used to differentiate between different parallel sessions.

Message (2) must require only minimal work for the Responder, since at that point he has no idea whether the Initiator is a legitimate correspondent or, *e.g.,* a forged message from an denial of service attack; no round trip has yet occurred with the Initiator. Therefore, it is important that the Responder not be required at this point to perform expensive calculations or create state. Here, the Responder's cost will be a single authentication operation, the cost of which (for HMAC) is dominated by two invocations of a cryptographic hash function, plus generation of a random nonce N_R.

The Responder *may* compute a new exponential $g^b \pmod{p}$ for each interaction. This is an expensive option, however, and at times of high load (or attack) it would be inadvisable. The nonce prevents two successive session keys from being the same, even if both the Initiator and the Responder are reusing exponentials.

If the Responder is willing to accept the group identified in the Initiator's message, his exponential must be in the same group. Otherwise, he may respond with an exponential from any group of his own choosing. The field $\mathrm{grpinfo}_R$ lists what groups the Responder finds acceptable, if the Initiator should wish to restart the protocol. This provides a simple mechanism for the Initiator to discover the groups currently allowed by the Responder. That field also lists what encryption algorithm is acceptable for the next message. This is not negotiated; the Responder has the right to decide what strength encryption is necessary to use his services.

Note that the Responder creates no state when sending this message. If it is fraudulent — that is, if the Initiator is non-existent or intent on perpetrating a denial-of-service attack — the Responder will not have committed any storage resources.

In Message (3), the Initiator echoes content from the Responder's message, including the authenticator. The authenticator allows the Responder to verify that he is in round-trip communication with a legitimate potential correspondent. The Initiator also uses the key derived from the two exponentials and the two nonces to encrypt her identity and service request. (The Initiator's nonce is used to ensure that this session key is unique, even if both the Initiator and the Responder are reusing their exponentials and the Responder has "forgotten" to change nonces.) The key used to protect Messages (3) and (4) is computed as $H_{g^{ir}}(N_I, N_R, 1)$. The session key used by IPsec (or any other application) is $H_{g^{ir}}(N_I, N_R, 0)$.

Because the Initiator can validate the Responder's identity before sending her own and because her identifying information (ignoring her public key signature) is sent encrypted, her privacy is protected from both passive and active attackers. (An active attacker can replay an old Message (2) as a response to the Initiator's initial message, but he cannot retrieve the Initiator's identity from Message (3) because he cannot complete the Diffie-Hellman computation.) The service request is encrypted, too, since disclosure of it might identify the requester. The Responder may wish to require a certain strength of cryptographic algorithm for certain services.

Upon successful receipt and verification of this message, the Responder has a shared key with a party known to be the Initiator. The Responder further knows what service the Initiator is requesting. At this point, he may accept or reject the request.

The Responder's processing on receipt of Message (3) requires verifying an authenticator and, if that is successful, performing several public key operations to verify the Initiator's signature and certificate chain. The authenticator (again requiring two hash operations) is sufficient defense against forgery; replays, however, could cause considerable computation. The defense against this is to cache the corresponding Message (4); if a duplicate Message (3) is seen, the cached response is retransmitted; the Responder does not create any new state or notify the application (*e.g.,* IPsec).

Caching Message (3) and refraining from creating new state for replayed instances of Message (3) serves also another security purpose. If the Responder were to create a new state and send a new Message (4), and a new sa' for a replayed Message (3), then an attacker that compromised the Initiator could replay a recent session with the Responder. That is, by replaying Message (3) from a recent exchange between the Initiator and the Responder, the attacker could establish a session with the Responder where the session-key is identical to the key of the previous session (which took place when the Initiator was not yet compromised). This could compromise the Forward Security of the Initiator.

There is a risk, however, in keeping this message cached for too long: if the Responder's machine is compromised during this period, perfect forward secrecy is compromised. We can tune this by changing the MAC key HK_R more frequently. The cache can be reset when a new g^r or HK_R is chosen.

In Message (4), the Responder sends to the Initiator any Responder-specific application data (*e.g.,* the Responder's IPsec SPI), along with a signature on both nonces, both exponentials, and the Initiator's identity. All the information is encrypted using a key derived the two nonces, N_I and N_R, and the Diffie-Hellman result. The Initiator can verify that the Responder is present and participating in the session, by decrypting the message and verifying the enclosed signature.

2.4 Rejection Messages

Instead of sending messages (2) or (4), the Responder can send a rejection instead. For Message (2), this rejection can only be on the grounds that he does not accept the group that the Initiator has used for her exponential. Accordingly, the reply should indicate what groups are acceptable. (For efficiency's sake, this information could also be in the Responder's long-lived certificate, which the Initiator may already have.)

Message (4) can be a rejection for several reasons, including lack of authorization for the service requested. But it could also be caused by the Initiator requesting cryptographic algorithms that the Responder regards as inappropriate, given the requester (Initiator), the service requested, and possibly other information available to the Responder, such as the time of day or the Initiator's location as indicated by the network. In these cases, the Responder's reply should list acceptable cryptographic algorithms, if any. The Initiator would then send a new Message (3), which the Responder would accept anew; again, the Responder does not create any state until after a successful Message (3) receipt.

3 What JFK Avoids

By intent, JFK does not do certain things. It is worth enumerating them, if only to forestall later attempts to add them in. The "missing" items were omitted by design, in the interests of simplicity.

The most obvious "omission" is any form of authentication other than certificate chain trusted by the each party. We make no provisions for shared secrets, token-based authentication, certificate discovery, or explicit cross-certification of PKIs. In our view, these are best accomplished by outboard protocols. Initiators that wish to rely on any form of legacy authentication can use the protocols being defined by the IPSRA [4] or SACRED [5, 6] working groups. While these mechanisms do add extra round trips, the expense can be amortized across many JFK negotiations. Similarly, certificate chain discovery (beyond the minimal capabilities implicit in ID_I and ID_R) should be accomplished by protocols defined for that purpose. By excluding the protocols for JFK, we can exclude them from our security analysis; the only interface between the two is a certificate chain, which by definition is a stand-alone secure object.

We also eliminate negotiation, in favor of ukases issued by the Responder. The responder is providing a service; it is entitled to set its own requirements for that service. Any cryptographic primitive mentioned by the Responder is acceptable; the Initiator can choose any it wishes. We thus eliminate complex rules for selecting the "best" choice from two different sets. We also eliminate state to be kept by the Responer; the Iniator can either accept the Responder's desires or restart the protocol.

Finally, we reject the notion of two different phases. As noted, the practical benefits of quick mode are limited. Furthermore, we do not agree that frequent rekeying is necessary. If the underlying block cipher is sufficiently limited as to bar long-term use of any one key, the proper solution is to replace that cipher. For example, 3DES is inadequate for protection of very high speed transmissions, because the probability of collision in CBC mode becomes too high after less than encryption of 2^{32} plaintext blocks. Using AES instead of 3DES solves that problem without complication the key exchange.

4 Related Work

4.1 Internet Key Exchange (IKE)

The Internet Key Exchange protocol (IKE) [1] is the current IETF standard for key establishment and SA parameter negotiation. IKE is based on the ISAKMP [7] framework, which provides encoding and processing rules for a set of payloads commonly used by security protocols, and the Oakley protocol, which describes an adaptation of the StS protocol for use with IPsec.

IKE is a two-phase protocol: during the first phase, a secure channel between the two key management daemons is established. Parameters such as an authentication method, encryption/hash algorithms, and a Diffie-Hellman group are negotiated at this point. This set of parameters is called a "Phase 1 SA." Using this information, the peers authenticate each other and compute key material using the Diffie-Hellman algorithm. Authentication can be based on public key signatures, public key encryption, or preshared passphrases. There are efforts to extend this to support Kerberos tickets [8] and handheld authenticators. It should also be noted that IKE can support other key establishment mechanisms (besides Diffie-Hellman), although none has been proposed yet[1].

Furthermore, there are two variations of the Phase 1 message exchange, called "main mode" and "aggressive mode." Main mode provides identity protection, by transmitting the identities of the peers encrypted, at the cost of three message round-trips . Aggressive mode provides somewhat weaker guarantees, but requires only three messages .

As a result, aggressive mode is very susceptible to untraceable[2] denial of service (DoS) attacks against both computational and memory resources [9]. Main mode is also susceptible to untraceable memory exhaustion DoS attacks, which must be compensated for in the implementation using heuristics for detection and avoidance. In particular:

– The Responder has to create state upon receiving the first message from the Initiator, since the Phase 1 SA information is exchanged at that point. This allows for a DoS attack on the Responder's memory, using random source IP addresses to send a flood of requests. To counter this, the Responder could employ mechanisms similar to those employed in countering TCP SYN attacks[10–12]. JFK avoids maintaining state at all as a result of receiving the first message.
– An Initiator who is willing to go through the first message round-trip (and thus identify her address) can cause the Responder to do a Diffie-Hellman exponential generation as well as the secret key computation on reception of the third message of the protocol. The Initiator could do the same with the fifth message of the protocol, by including a large number of bogus certificates, if the Responder blindly verifies all signatures. JFK mitigates the effects of this attack by reusing the same exponential across different sessions.

The second phase of the IKE protocol is commonly called "quick mode" and results in IPsec SAs established between the two negotiating parties, through a three-message exchange. Parameters such as the IP security protocol to use (ESP/AH), security algorithms, the type of traffic that will be protected, *etc.* are negotiated at this stage. Since the two parties have authenticated each other and established a shared key during Phase 1, quick mode messages are encrypted and authenticated using that information. Furthermore, it is possible to derive the IPsec SA keying material from the shared key established during the Phase 1 Diffie-Hellman exchange. To the extent that multiple IPsec SAs between the same two hosts are needed, this two-phase approach results in faster and more lightweight negotiations (since the same authentication information and keying material is reused).

Unfortunately, two hosts typically establish SAs protecting all the traffic between them, limiting the benefits of the two-phase protocol to lightweight

[1] There is ongoing work (still in its early stages) in the IETF to use IKE as a transport mechanism for Kerberos tickets, for use in protecting IPsec traffic.
[2] The attacker can use a forged address when sending the first message in the exchange.

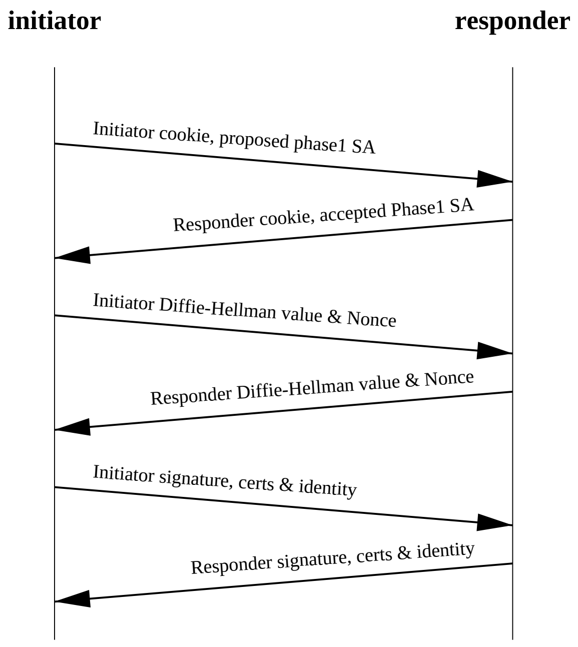
initiator
responder
Initiator cookie, proposed phase1 SA
Responder cookie, accepted Phase1 SA
Initiator Diffie-Hellman value & Nonce
Responder Diffie-Hellman value & Nonce
Initiator signature, certs & identity
Responder signature, certs & identity

Fig. 1. IKE Main Mode exchange with certificates.

re-keying. If "Perfect Forward Secrecy" (PFS) is desired, this benefit is further diluted. (PFS is an attribute of encrypted communications allowing for a long-term key to be compromised without affecting the security of past session keys.)

Another problem of the two-phase nature of IKE manifests itself when IPsec is used for fine-grained access control to network services. In such a mode, credentials exchanged in the IKE protocol are used to authorize users when connecting to specific services. Here, a complete Phase 1 & 2 exchange will have to be done for each connection (or, more generally, traffic class) to be protected, since credentials, such as public key certificates, are only exchanged during Phase 1.

IKE protects the identities of the Initiator and Responder from eavesdroppers[3]. The "identities" include public keys, certificates, and other information that would allow an eavesdropper to determine which principals are trying to communicate. These identities can be independent of the IP addresses of the IKE daemons that are negotiating (*e.g.,* temporary addresses acquired via DHCP, public workstations with smartcard dongles, *etc.*). However, since the Initiator reveals her identity first (in message 5 of Main Mode), an attacker can pose as the Responder until that point in the protocol. The attacker cannot complete the protocol (since they do not possess the Responder's private key), but they can determine the Initiator's identity. This attack is not possible on the Responder,

[3] Identity protection is provided only in Main Mode (also known as Identity Protection Mode); Aggressive Mode does not provide Identity Protection for the Initiator.

since she can verify the identity of the Initiator before revealing her identity (in message 6 of Main Mode). However, since most Responders would correspond to servers (firewalls, web servers, *etc.*), the identity protection provided to them seems not as useful as protecting the Initiator's identity[4]. Fixing the protocol to provide identity protection for the Initiator would involve reducing it to 5 messages and having the Responder send the contents of message 6 in message 4, with the positive side-effect of reducing the number of messages, but breaking the message symmetry and protocol modularity.

Finally, thanks to the desire to support multiple authentication mechanisms and different modes of operation (Aggressive *vs.* Main mode, Phase 1 / 2 distinction), both the protocol specification and the implementations tend to be bulky and fairly complicated. These are undesirable properties for a critical component of the IPsec architecture.

[13] points out many deficiencies in the IKE protocol, specification, and implementation. It suggests removing several features of the protocol (*e.g.,* aggressive mode, public key encryption mode, *etc.*), restore the idea of stateless cookies, and protect the Initiator's (instead of the Responder's) identity from an active attacker. It also suggests some other features, such as one-way authentication (similar to what is common practice when using SSL[14, 15] on the web). These major modifications would bring the IKE protocol closer to JFK, although they would not completely address the DoS issues.

A measure of the complexity of IKE can be found in the analyses done in [16, 17]. No less than 13 different sub-protocols are identified in IKE, making understanding, implementation, and analysis of IKE challenging. While the analysis did not reveal any attacks that would compromise the security of the protocol, it did identify various potential attacks (DoS and otherwise) that are possible under some *valid* interpretations of the specification and implementation decisions.

4.2 Other Protocols

The predecessor to IKE, Photuris[2], first introduced the concept of cookies to counter "blind" denial of service attacks. The protocol itself is n 6-message variation of the Station to Station protocol. It is similar to IKE in the message layout and purpose, except that the SA information has been moved to the third message. For re-keying, a two-message exchange can be used to request a unidirectional SPI (thus, to completely re-key, 4 messages are needed). Photuris is vulnerable to the same computation-based DoS attack as IKE, mentioned above.

SKEME[18] shares many of the requirements for JFK, and many aspects of its design were adopted in IKE. It serves more as a set of protocol building blocks, rather than a specific protocol instance. Depending on the specific requirements for the key management protocol, these blocks could be combined in several ways. As a result of this modularization, both the number of round-trips and

[4] One case where protecting the Responder's identity can be more useful is in peer-to-peer scenarios.

the optional payloads and exchanges is quite high. The latter has a direct impact on the implementation complexity (as seen in IKE itself). Another interesting aspect of SKEME is its avoidance of digital signatures; public key encryption is used instead, to provide authentication assurances. The reason behind this was to allow both parties of the protocol to be able to repudiate the exchange.

SKIP[19] was an early proposal for an IPsec key management mechanism. It uses long-term Diffie-Hellman public keys to derive long-term shared keys between parties, which is used to distribute session keys between the two parties. The distribution of the session key occurs in-band, *i.e.,* the session key is encrypted with the long-term key and is injected in the encrypted packet header. While this scheme has good synchronization properties in terms of re-keying, it lacks any provision for PFS. Furthermore, there is no identity protection provided, since the certificates used to verify the Diffie-Hellman public keys are (by design) publicly available, and the source/destination master identities are contained in each packet (so a receiver can retrieve the sender's Diffie-Hellman certificate). The latter can be used to mount a DoS attack on a receiver, by forcing them to retrieve and verify a Diffie-Hellman certificate, and then compute the Diffie-Hellman shared secret.

References

1. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force (1998)
2. Karn, P., Simpson, W.: Photuris: Session-key management protocol. Request for Comments 2522, Internet Engineering Task Force (1999)
3. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: keyed-hashing for message authentication. Request for Comments 2104, Internet Engineering Task Force (1997)
4. Sheffer, Y., Krawczyk, H., Aboba, B.: PIC, a pre-IKE credential provisioning protocol. Internet Draft, Internet Engineering Task Force (2001) Work in progress.
5. Arsenault, A., Farrell, S.: Securely available credentials - requirements. Request for Comments 3157, Internet Engineering Task Force (2001)
6. Gustafson, D., Just, M., Nystrom, M.: Securely available credentials - credential server framework. Internet Draft, Internet Engineering Task Force (2001) Work in progress.
7. Maughan, D., Schertler, M., Schneider, M., Turner, J.: Internet security association and key management protocol (ISAKMP). Request for Comments (Proposed Standard) 2408, Internet Engineering Task Force (1998)
8. Miller, S.P., Neuman, B.C., Schiller, J.I., Saltzer, J.H.: Kerberos Authentication and Authorization System. Technical report, MIT (1987)
9. Simpson, W.A.: IKE/ISAKMP Considered Harmful. USENIX ;login: (1999)
10. Heberlein, L., Bishop, M.: Attack Class: Address Spoofing. In: Proceedings of the 19th National Information Systems Security Conference. (1996) 371–377
11. CERT: Advisory CA-96.21: TCP SYN Flooding. ftp://info.cert.org/pub/cert_advisories/CA-96.21.tcp_syn_flooding (1996)
12. Schuba, C., Krsul, I., Kuhn, M., Spafford, E., Sundaram, A., Zamboni, D.: Analysis of a denial of service attack on tcp. In: IEEE Security and Privacy Conference. (1997) 208–223

13. Kaufman, C., et al.: Code-preserving Simplifications and Improvements to IKE. Internet Draft, Internet Engineering Task Force (2001) Work in progress.
14. Hickman, K.: Secure Socket Library (SSL). http://home.netscape.com/security/techbriefs/ssl.html (1995)
15. Dierks, T., Allen, C.: The TLS protocol version 1.0. Request for Comments (Proposed Standard) 2246, Internet Engineering Task Force (1999)
16. Meadows, C.: Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In: Proc. of the 1999 IEEE Symposium on Security and Privacy. (1999) 216–231
17. Meadows, C.: Open issues in formal methods for cryptographic protocol analysis. In: Proc. of DARPA Information Survivability Conference and Exposition (DISCEX 2000), IEEE Computer Society Press (2000) 237–250
18. Krawczyk, H.: SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In: Proc. of Network and Distributed System Security Symposium (NDSS). (1996)
19. Aziz, A., Patterson, M.: Simple Key Management for Internet Protocols (SKIP. In: Proc. of the 1995 INET conference. (1995)