Network Working Group
Internet Draft
draft-ietf-ipsec-jfk-04.txt
Expires in 6 months

W. Aiello
S.M. Bellovin
M. Blaze
R. Canetti
J. Ioannidis
A.D. Keromytis
O. Reingold

Just Fast Keying (JFK)

Abstract

This draft discusses JFK, a key management protocol.

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 [RFC2409], 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 (in no particular order):

- Security: The resulting key should be cryptographically secure, according to standard measures of cryptographic security for key-exchange.

- 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 provide protection for the privacy of the parties.
  (Different variants are considered within.)

- 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. 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.

Let us highlight two ways in which the simplicity requirement is met by the proposed protocol. First, we avoid the complex two-phase structure of IKE. This buys us a great deal in terms of simplicity, with little to no cost in terms of functionality. (See more details within.) Second, we restrict the protocol to a small and fixed number of rounds: two rounds trips, always, with no optional additional rounds. This results in greater simplicity in implementation, debugging, customization, and operation.

The Memory-DoS and Computation-DoS properties have become more important in the context of recent Internet denial-of-service attacks. Photuris [RFC2522] 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. Photuris first introduced the concept of cookies to counter

``blind'' denial of service attacks. Although the concept of the cookie was adopted by IKE, its use in that protocol did not follow the guidelines established by Photuris and left it open to DoS attacks.

The Privacy property means that the protocol does not reveal the identities of the parties to an attacker. There are several variants here: First, the protection can cover the initiator, or the responder or both. Second, the protection can be valid either against active attackers or alternatively only against passive eavesdroppers. We present two protocol variants: One variant provides identity protection for the initiator against active attacks, and no protection for the responder. This variant seems more suitable for a client-server scenario where the initiator (the client) needs full identity protection while the responder (the server) needs no identity protection. The other variant protects the identity of the responder from active attackers, and in addition protects the identity of the initiator from eavesdroppers. This variant seems more suitable for peer-to-peer scenarios where both parties need identity protection but the responder is more vulnerable to active attacks. We leave it to the working group to choose the variant that best suites the needs of IPSEC. (Potentially, one could design a protocol where the parties negotiate whose identity needs to be protected against active attackers. However we believe this would result in an unnecessarily complex protocol.)

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-Negotiation property is necessary for several reasons. The first, of course, is as a corollary to Simplicity 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. JFK 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.

We present two protocols (or, rather, two variants of the protocol). The variants, denoted JFKi and JFKr, are very similar in many respects, with two main differences: JFKi provides active identity protection for the initiator and no identity protection for the responder, whereas JFKr provides active identity protection for the responder and passive identity protection for the initiator. In addition, JFKi contains an additional (amortizable) signature. We note that the cryptographic core of JFKr follows that of SIGMA [K01] and IKEv2 [IKEv2].

2. The JFK Protocol

2.1 Notation

E{K}(M): encryption of M with symmetric key K.

HMAC{K}(M): keyed hash of M using key K in an HMAC scheme [RFC2104].

SIG{x}(M): digital signature of M using the private key belonging to principal x (Initiator or Responder). It is not assumed to be a message-recovering signature (but it can be).

The message components used in JFK are as follows:

g^x: Diffie-Hellman exponentials, also identifying the group-ID. The Diffie-Hellman groups identified in [RFC2409] are used.

g^i, g^r: Initiator and Responder exponentials.

Ni: Initiator nonce, a random bit-string. The Initiator MUST pick a fresh nonce at each invocation of the JFK protocol. The first few bytes (2 or 4, for IPv4 or IPv6 respectively) of this nonce may contain a cookie used by the Responder to help the Initiator avoid fragmentation-based DoS attacks (see Appendix B).

IPi: The Initiator's network identifier (IPv4 or IPv6 address). It is used by the Responder to counter a "cookie-jar" attack, when verifying the authenticator upon receipt of Message 3.

Nr: Responder nonce, a random bit-string. The Responder MUST pick a fresh nonce at each invocation of the JFK protocol. The nonces are used in the session key computation, to provide key independence when one or both parties reuse the same

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. The first few bytes (2 or 4, for IPv4 or IPv6 respectively) of this nonce may contain a cookie used by the Initiator to help the Responder avoid fragmentation-based DoS attacks (see Appendix B).

sa: Defines the cryptographic and other properties of the Security Association (SA) the Initiator wants to establish. It contains a Domain-of-Interpretation, which JFK understands, and an application-specific bit-string.

sa': Any information that the Responder needs to provide to the Initiator with respect to the application SA (e.g., the Responder's SPI, in IPsec).

HKr: A transient hash key private to the Responder; this is a global parameter for the Responder (i.e., it is not different for every different protocol run), which changes periodically: the Responder must pick a new g^r every time HKr changes. The use of HKr and the implications of changing it periodically will be explained later in this section.

Kir: A shared key derived from g^ir, Ni, and Nr, used as part of the application SA (e.g., IPsec SA).

Ke: A shared key derived from g^ir, Ni, and Nr, used to protect the secrecy of messages 3 and 4 of the protocol. Although the input parameters are the same with Kir, a different key derivation mechanism is used to ensure key independence.

Ka: A shared key derived from g^ir, Ni, and Nr, used to integrity-protect messages 3 and 4 of the protocol. Although the input parameters are the same with Kir and Ke, a different key derivation mechanism is used to ensure key independence.

Ks: A shared key derived from g^ir, Ni, and Nr, used to authenticate subsequent runs of the protocol. Although the input parameters are the same with Kir, Ke, and Ka, a different key derivation mechanism is used to ensure key independence.

IDi, IDr: Initiator and Responder certificates, or public-key identifying information, or previous-state identifying information. Multiple such payloads may appear in a message, to indicate multiple certificates, CRLs, etc. For simplicity and clarity, the notation in this draft shows only one such payload per message.

IDr': an indication by the Initiator to the Responder as to what identity (and corresponding key material) the latter should use to authenticate to the former. The Responder may ignore this hint. This field may contain the certificate of a CA trusted by the Initiator (which means that the Initiator is requesting that the Responder authenticate with a certificate chain "rooted" at that CA), or the certificate of the Responder (effectively identifying the public, and corresponding private,

key of the Responder).

GRPINFOr: A list of all Diffie-Hellman groups supported by the Responder, the symmetric algorithm used to protect messages 3 and 4, and the hash function used for key generation.

2.2 Protocol JFKr

This variant of the JFK protocol provides the same DoS protection and identity protection against passive attackers for both the Initiator and the Responder, but no protection against active identity discovery attacks for the Initiator (the Responder is protected against active identity discovery).

Using the same notation as in section 2, the JFKr protocol is:

Message 1, I->R: Ni, g^i

Message 2, R->I: Ni, Nr, g^r, GRPINFOr,
HMAC{HKr}(g^r, Nr, Ni, IPi)

Message 3, I->R: Ni, Nr, g^i, g^r, HMAC{HKr}(g^r, Nr, Ni, IPi),
E{Ke}(IDi, IDr' sa, SIG{i}(Ni, Nr, g^i, g^r, GRPINFOr)),
HMAC{Ka}('I', E{Ke}(IDi, IDr', sa, SIG{i}(Ni, Nr, g^i, g^r, GRPINFOr)))

Message 4, R->I: Ni, Nr, E{Ke}(IDr, sa', SIG{r}(g^r, Nr, g^i, Ni)),
HMAC{Ka}('R', E{Ke}(IDr, sa', SIG{r}(g^r, Nr, g^i, Ni)))

The keys used to protect Messages (3) and (4), Ke and Ka, are computed as HMAC{g^ir}(Ni, Nr, 1) and HMAC{g^ir}(Ni, Nr, 2) respectively. The session key used by IPsec (or any other application), Kir, is HMAC{g^ir}(Ni, Nr, 0). Key Ks, used for lightweight mode authentication (see Section 3), is computed as HMAC{g^ir}{Ni, Nr, 3}.

If more key material is needed, the same mechanism for "key stretching" as specified in [IKEv2] may be used.

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.

In Message (2), the Responder replies with his own exponential (in the same group), information on what secret key algorithms are acceptable for the next message, a random nonce, and an authenticator calculated from a secret, HKr, known to the Responder; the authenticator is computed over the Responder exponential, the two nonces, and the Initiator's network address. The Responder's exponential may also be reused; again, it is

regenerated according to the Responder's forward secrecy interval. 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 authenticator also confirms that the sender of Message (3) uses the same address as in Message (1) --- this can be used to detect and counter a ``cookie jar'' DDoS attack. 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 Ni and Nr. The encryption and authentication use algorithms specified in GRPINFOr. 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 (or other application using JFKr as the key establishment protocol). This cache of messages can be reset as soon as HKr is 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). Note: It is important that the responder deals 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. The message is also protected by an MAC (such as HMAC), using a key derived from the Diffie-Hellman computation and the nonces Ni and Nr. Notice that this key is different form the one used for encrypting the message.

Message (4) contains application-specific information (such as the Responder's IPsec SPI), and a signature on both nonces and both exponentials. Everything is encrypted by Ke, which is derived from Ni, Nr, and g^ir (the result of the Diffie-Hellman computation). As in Message (3), this message is also protected by a MAC using key Ka.

We remark that the core cryptographic design of JFKr follows that of SIGMA [K01] and IKEv2 [IKEv2].

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 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 protect both parties from revealing their identities in the clear. An active attacker can determine the Initiator's identity by performing a man-in-the-middle attack, since the Initiator must

authenticate to the Responde first (this is a consequence of the 4-message exchange combined with the "liveness" requirement for DDoS prevention).

The Initiator's initial message, Message (1), is a straight-forward 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 Ni 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 Nr. Notice that, if the Responder is reusing the same g^r across multiple Initiators, the beginning of the HMAC computation can be cached along with g^r.

The Responder may compute a new exponential g^r 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.

A simple way of dealing with DDoS is to periodically (e.g., once every 30 seconds) generate an (r, g^r, HMAC(HKr){g^r}) tuple and place it in a FIFO queue. As requests arrive (in particular, as valid Message (3)'s are processed), the first entry from the FIFO is removed. If the rate of requests exceeds the generating rate, a JFKr implementation should reuse the last tuple in the FIFO. Notice that in this scheme, the same g^r may be reused in different sessions, if those sessions are interleaved. This does not violate the PFS or other security properties of the protocol.

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 GRPINFOr 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 bogus --- 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. She 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), Ke, is computed as HMAC{g^ir}{Ni, Nr, 1}. The session key used by IPsec (or any other application), Kir, is HMAC{g^ir}(Ni, Nr, 0). The message authentication key, Ka, is computed as HMAC{g^ir}{Ni, Nr, 2}. For this computation, the values '0', '1', and '2' correspond to a single-byte decimal value (so the HMAC computation is over the two nonces and one extra byte, with the corresponding value).

In JFKr, the parties obtain a shared encryption key, Ke, before any of the parties send their identities. Therefore, both parties send their identities encrypted with Ke, thus providing both parties with identity protection against passive eavesdroppers. In addition, the party that first reveals its identity is the initiator. This way, the responder is required to reveal its identity only after it verifies the identity of the initiator. This guarantees active identity protection to the responder.

The service request (sa payload) is encrypted too, since its disclosure might identify the requester. The Responder may wish to require a certain strength of cryptographic algorithm for certain services.

We remark that it is essentially impossible, using current technology, to provide a two-round-trip protocol that provides DOS protection for the responder, passive identity protection for both parties, and active identity protection for the initiator. An informal argument proceeds as follows: If DoS protection is in place, then the responder must be able to send his first message before he computes any shared key. (This is so since computing a shared key is a relatively costly operation in current technology.) This means that the responder cannot send its identity in the second message, without compromising its identity protection against passive eavesdroppers. This means that the responder's identity must be sent in the fourth (and last) message of the protocol. Consequently, the initiator's identity must be sent before the responder's identity is sent.

Upon successful receipt and verification of message (3), 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). The key for looking up Message 3's in the cache is the authenticator; this prevents DoS attacks where the attacker randomly modifies the encrypted blocks of a valid message, causing a cache miss and thus more processing to be done at the Responder. Further, if the authenticator verifies but there is some problem with the message (e.g., the certificates do not verify), the responder can cache the authenticator along with an indication as to the failure (or the actual rejection message), to avoid unnecessary processing (which may be part of a DoS attack). This cache of Message(3)'s and authenticators can be purged as soon as HKr is changed (since the authenticator will no longer pass verification).

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, to 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 HKr more frequently. The cache can be reset when a new HKr 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, Ni and Nr, 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. Since Message (2) already

contains the field GRPINFOr (which indicates what groups are acceptable), no explicit rejection message is needed. (For efficiency sake, the group 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 de novo; again, the Responder does not create any state until after a successful Message (3) receipt.

2.5 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.

In our view, any form of authentication other than certificate chain trusted by the other party is 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 or SACRED 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 IDi and IDr) should be accomplished by protocols defined for that purpose. By excluding these 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. However, it is possible to use the lightweight JFK mode, as described in Section 3, to perform shared-secret based authentication.

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 Responder; the Initiator can either accept the Responder's desires or restart the protocol.

Finally, we reject the notion of two different phases. 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. Should low-cost rekeying be necessary, the JFK protocol itself can be run in ``lightweight'' mode, as we describe in Section 3.

2.6 Phase II and the Lack Thereof

Phase II of IKE is used for several things. We do not regard any of them as necessary.

One is generating the actual keying material used for security associations. It is expected that this will be done several times, to amortize the expense of the Phase I negotiation. A second reason for this is to permit very frequent rekeying. Finally, it permits several separate security associations to be set up, with different parameters.

We do not think these apply. First, with modern ciphers, such as AES, there is no need for frequent key changes. AES keys are long enough that brute force attacks are infeasible. Its longer block size protects against CBC limitations when encrypting many blocks [BDJR97].

We also feel that JFK is efficient enough that avoiding the overhead of a full key exchange is not required. Rather than add new SAs to an existing Phase I SA, we suggest that a full JFK exchange be initiated instead. We note that the initiator can also choose to reuse its exponential, it if wishes to trade perfect forward secrecy for computation time. If state already exists between the initiator and the responder, they can simply check that the Diffie-Hellman exponentials are the same; if so, the result of the previous exponentiation can be reused. As long as one of the two parties uses a fresh nonce in the new protocol exchange, the resulting cryptographic keys will be fresh and not subject to a related key (or other, similar) attack. As we discuss in Section 3, a similar performance optimization can be used on the certificate chain validation.

A second major reason for Phase II is dead peer detection. IPsec gateways often need to know if the other end of a security association is dead, both to free up resources and to avoid "black holes". In JFK, this is done by noting the time of the last packet received. A peer that wishes to elicit a packet may send a "ping". Such hosts MAY decline any proposed security associations that do not permit such "ping" packets.

A third reason for Phase II is general security association control, and in particular SA deletion. While such a desire is not wrong, we prefer not to burden the basic key exchange mechanism with extra complexity. There are a number of possible approaches. Our requires that JFK endpoints implement the following rule: a new negotiation that specifies an SPD identical to the SPD of an

existing SA overwrites it. To some extent, this removes any need to delete an SA if black hole avoidance is the concern; you simply negotiate a new one. If you wish to delete an SA without replacing it, negotiate a new SA with the ESP_BYPASS, AH_BYPASS or IPCOMP_BYPASS ciphersuite.

3. Rekeying

When a negotiated SA expires (or shortly before it does), the JFK protocol is run again. It is up to the application to select the appropriate SA to use among many valid ones. In the case of IPsec, implementations should switch to using the new SA for outgoing traffic, but would still accept traffic on the old SA (as long as that SA has not expired).

To address performance considerations, we should point out that, properly implemented, rekeying only requires one signature and one verification operation in each direction, if both parties use the same Diffie-Hellman exponentials (in which case the cached result can be reused) and certificates: the receiver of an ID payload compares its hash with those of any cached ID payloads received from the same peer. While this is an implementation detail, a natural location to cache past ID payloads is along with already established SAs (a convenient fact, as rekeying will likely occur before existing SAs are allowed to expire --- so the ID information will be readily available). If a match is found and the result has not "expired" yet, then we do not need to re-validate the certificate chain. A previously verified certificate chain is considered valid for the shortest of its CRL re-validate time, certificate expiration time, OCSP result validity time, etc. For each certificate chain, there is one such value associated (the time when one of its components becomes invalid or needs to be checked again).

Notice that an implementation does need to cache the actual ID payloads; all that is needed is the hash and the expiration time.

That said, if for some reason fast rekeying is needed for some application domain, it can be done by re-using the JFKr protocol itself, using an ID payload identifying a previous exchange (and thus a corresponding secret key generated during that exchange) and an HMAC instead of a public-key signature for authentication.

```
Message 1, I->R:  Ni, g^i

Message 2, R->I:  Ni, Nr, g^r, GRPINFOr,
                  HMAC{HKr}(g^r, Nr, Ni, IPi)

Message 3, I->R:  Ni, Nr, g^i, g^r, HMAC{HKr}(g^r, Nr, Ni, IPi),
              E{Ke}(IDi, IDr' sa, HMAC{Ks}(Ni, Nr, g^i, g^r,
GRPINFOr)),
    HMAC{Ka}('I', E{Ke}(IDi, IDr', sa, HMAC{Ks}(Ni, Nr, g^i, g^r,
GRPINFOr)))

Message 4, R->I:  Ni, Nr, E{Ke}(IDr, sa', HMAC{Ks}(g^r, Nr, g^i,
```

Ni)),
HMAC{Ka}('R', E{Ke}(IDr, sa', HMAC{Ks}(g^r, Nr, g^i, Ni)))

The only difference from the basic JFKr protocol is the use of an HMAC in messages 3 and 4 for peer authentication. The HMAC is computed over the same fields as the public key signature in the basic protocol. The shared key, Ks, is either statically configured (when JFKr is used for shared-secret based authentication) or is derived from a previous exchange.

Payloads IDi and IDr identify the secret key in use. For shared-secret authentication, these are free-form strings that are configured by the administrator (or otherwise provided through means outside the JFK protocol's scope).

When Ks is derived from a previous JFKr exchange, the IDi and IDr payloads contain unique identifiers provider by the Responder and the Initiator respectively in the previous protocol exchange, and are used to identify the shared-secret. These identifiers should be treated as opaque bitstrings by the receiver. Effectively, at each protocol run, each party provides their peer with a cookie. This cookie can be used by the peer in subsequent rounds to identify the shared secret to use for authentication.

When run in lightweight mode, and if both parties reuse the exponentials g^i and g^r, new application (e.g., IPsec) SAs can be established without any public-key operations. If PFS is desired, then both parties have to perform one modular exponentiation operation (for the Diffie-Hellman computation). No signature or certificate verification is require for this mode.

4. Wire Format

This section describes a proposal for the specific protocol elements for the protocol described in this document. The authors of the document are not strongly attached to these proposed elements. More detail on the protocol elements will be added in later drafts.

The protocol will be run over UDP on a port to be assigned later by IANA. UDP is chosen to avoid well-known TCP attacks, although running JFK over UDP may cause some problems with packet fragmentation and reordering. For pre-standards testing purposes, UDP port 1024 which is reserved by IANA and will *not* be the eventual port for JFK.

Implementors of IKE have long complained that the specification required or strongly suggested too many algorithms that had essentially the same properties. Because of this, JFK only lists one option for each type of algorithm below. In the future, additional options might be added (which is why there are algorithm identifiers in the protocol), but they should only be added if there is a strong security requirement for them. Two such requirements would be the compromise of one of the listed algorithms or the adoption of a much stronger or much more capable

algorithm. Additional algorithms can only be added by a standards-track RFC.

4.1 Structure

Each message is a string of tag-length-value elements concatenated together. Tags are one octet. Lengths are two octets, and specify the number of octets of the value. Values are always integral numbers of octets. All octets are in big-endian order.

The values for the tags are:

Tag	Value (in decimal)
Ni	1
Nr	2
g^i	3
g^r	4
GRPINFOr	5
IDi	6
IDr, IDr'	7
Signature	8
HashedInfo	9
encrypt_i	10
encrypt_r	11
sa, sa'	12
rejectinfo_to_msg3	13

4.2 Description of the values for each tag

Nonces Ni and Nr MUST be 8 octets or longer.

g^i and g^r are expressed as a single octet specifying the group number, followed by value of Diffie-Hellman exponential. The group number is the same as the group numbers used in [RFC2409].

GRPINFOr is expressed as a string of at least four octets. The first octet is the encryption algorithm ID, the second octet is the signature algorithm ID, and the third octet is the hash function used for session key derivation. Each remaining octet specifies an acceptable group number.

IDi, IDr, and IDr' are expressed as a single octet specifying the type of ID used, followed by the ID material. The following ID types are specified.

ID tag	Meaning
1	PKIX certificate
2	CRL
3	OCSP response

Signatures are expressed as one octet specifying the signature algorithm followed by the octets of the signature.

HashedInfo is expressed as one octet specifying the keyed hash

algorithm followed y the octets of the hash.

encrypt_i and encrypt_r are expressed as one octet specifying the encryption algorithm followed by the encrypted content. When using a block cipher with chaining, requiring an Initialization Vector (IV), the IV is prepended to the ciphertext and constitutes the first block of the payload.

sa and sa' are expressed by one octet specifying the SA type followed by the SA itself. The following SA types are specified.

SA tag	Meaning
1	IPsec SA, as described below.
2	Opaque identifier for lightweight mode authentication. Contents should be treated as a bit-string and passed as is to the peer on subsequent rounds of the protocols, to identify the correct key Ks to use for the HMAC computation.

Both types (1 and 2) of sa payload may appear in messages 3 and 4.

rejectinfo_to_msg3 has the same structure as grpInfo.

Encryption algorithm IDs (for use in JFK only)

3DES	1

Signature algorithm IDs (for use in JFK only)

RSA	1

Hash algorithm IDs (for use in JFK only)

SHA-1	1

5. Security Associations and the Security Policy Database

Security Association in JFK follows the two primary principles of JFK: simplicity and lack of negotiation. The latter is straightforward: an initiator proposes an SA; the responder may either accept it or reject it with a reason.

The myriad combinations of cipher and authentication suites available in IKE contributed in no small measure to interoperability problems. We follow the suggestion put forth by an early IKEv2 draft: the acceptable combinations are denoted by 16-bit, unstructured integers. The integers may happen to be assigned according to some structuring principle, but implementations MUST NOT treat them as other than opaque values.

The initial set of algorithms that MUST be supported is:

- ESP-AES-CBC with HMAC-SHA1
- ESP-3DES-CBC with HMAC-MD5
- ESP-3DES-CBC with HMAC-SHA1
- ESP-NULL with HMAC-MD5
- ESP-NULL with HMAC-SHA1
- ESP_BYPASS

```
    AH with HMAC-MD5
    AH with HMAC-SHA1
    AH_BYPASS
    IPCOMP_DEFLATE
    IPCOMP_BYPASS
```

The 3DES choices are for backwards compatibility. We do not specify a DES variant -- DES was demonstrated to be insecure several years ago, and implementors have had time to upgrade. We do support MD5, but recommend that it not be used.

The *_BYPASS entries are used to indicate that existing SAs of that type that have previously been negotiated between the two peers should be deleted.

Following the algorithm choice is a 4-byte value containing the sender's SPI, in network byte order (big endian format).

Following the SPI are two lists of zero or more SPD elements, the list of source addresses followed by the list of destination addresses.

JFK supports two different address types: a list of IPv4 address ranges, and a list of IPv6 address ranges. A single address is a range where the starting and ending elements are the same. Subnets are converted to range format. The notion of "all addresses" is expressed as the pair (0, 0xFFFFFFFF) for IPv4; v6 is handled similarly.

Port numbers are also expressed as pairs; again, the concept of "all ports" is represented as (0, 0xFFFF). The same is done for protocols.

An SPD element has the following format:

```
    Address family version v4/v6 (2 bytes)
    Transport protocol range (TCP, UDP, etc.) (2 bytes)
    Number of address ranges (2 bytes)
            address ranges...
    Number of port number ranges (2 bytes)
            port number ranges...
```

An SPD source or destination specification consists of a two-byte counter for the number of SPD elements, followed by each element. The simplest entry -- all protocols, all ports, all addresses, for IPv4 -- would look like this:

```
    Bytes
     1 2
    ----
    0001    Number of SPD elements (1 element)
    0004    Address family (IPv4 addresses)
    00FF    Protocol range (all transport protocols)
    0001    One address range
    0000,0000  (4 bytes, 0.0.0.0)
    FFFF,FFFF  (4 bytes, 255.255.255.255)
```

```
0001    Number of port ranges (1 port range)
0000    Beginning of transport protocol port range (port 0)
FFFF    End of port range (port 65535)
```

Two specifications are contained in the sa and sa' payloads, one for the source range followed by one for the destination range.

When transmitted on the wire, all values are sent in big endian format.

6. Security Considerations

This section very briefly overviews our security analysis of the JFK protocol. Full details are deferred to the full analysis paper.

In general, there are currently two main approaches to analyzing security of protocols. One is the formal-methods approach, where the cryptographic components of a protocol are modeled by "ideal boxes" and automatic theorem-verification tools are used to verify the validity of the high-level design (assuming ideal cryptography). The other is the cryptographic approach, which accounts for the facts that cryptographic components are imperfect and may potentially interact badly with each other. Here security of protocols is proven based on some underlying computational intractability assumptions (such as the hardness of factoring large numbers, or computing discrete logarithms modulo a large prime, or inverting a cryptographic hash function). The formal-methods approach, being automated, has the advantage that it is less susceptible to human errors and oversights in analysis. On the other hand, the cryptographic approach provides better soundness since it considers the overall security of the protocol, and in particular accounts for the imperfections of the cryptographic components.

Our analysis follows the cryptographic approach. We welcome any additional analysis. In particular, analysis based on formal methods would be a useful complement to the analysis described here.

We separate the analysis of the "core security" of the protocol (which is rather tricky) from the analysis of added security features such as DoS protection and identity protection (which is much more straightforward). The rest of this section concentrates on the "core security" of the protocol. DoS and Identity protection were discussed in previous sections.

6.1 Core security

We use the modeling and treatment of [CK01], which in turn is based on [BR93]. See there for more references and comparisons with other analytical work. Very roughly, the "core security" of a key exchange protocol boils down to two requirements:

* If party A generates a key Ka associated with a

session-identifier S and peer identity B, and party B generates
a key Kb associated with the same session identifier S and peer
A, then Ka=Kb.

* No attacker can distinguish between the key exchanged in a session
between two unbroken parties and a truly random secret. This holds
even if the attacker has total control over the communication, can
invoke multiple sessions, and is told the keys generated in all
other sessions.

We stress that this is only a rough sketch of the requirement. For full details see [CK01,CK02]. We show that both JFKi and JFKr satisfy the above requirement. When these protocols are run with perfect forward secrecy, the security is based on a standard intractability assumption of the DH problem. When a party reuses its DH value, the security is based on a stronger assumption intractability assumption involving both DH and the HMAC pseudo random function.

We first analyze the protocols in the restricted case where the parties do not re-use the private DH exponents for multiple sessions.
(This is the bulk of the work.) Here the techniques for demonstrating
the security of the two protocols are quite different. Specifically:

(I) JFKi: The basic cryptographic core of this protocol is the same as
the ISO 9798-3 protocol, which was analyzed and proven secure in [CK01]. This protocol can be briefly summarized as follows:

A->B: A,N_a,g^a
B->A: B,N_b,g^b,SIG_b(N_a,N_b,g^a,g^b,A)
A->B: SIG_a(N_a,N_b,g^a,g^b,B)

A salient point about this protocol is that each party signs, in addition to the nonces and the two public DH exponents, also the identity of the peer. (If the peer's identity is not signed then the protocol is completely broken.) JFKi inherits the same basic core
security. In addition, JFKi adds a preliminary cookie mechanism for DoS protection (which results in adding one flow to the protocol and having the *responder* in JFKi play the role of A), and encrypts the last two messages in order to provide identity protection to the initiator.

(II) JFKr: The basic cryptographic core of this protocols is the same as that of the signature mode of IKE and SIGMA [K01], which was analyzed and proven secure in [CK02]. This basic protocol can be briefly summarized as follows:

A->B: N_a,g^a
B->A: B,N_b,g^b,SIG_b(N_a,N_b,g^a,g^b),MAC_{Ka}(Na,Nb,B)

A->B: A,SIG_a(N_a,N_b,g^a,g^b),MAC_{Ka}(Na,Nb,A)

Here neither party signs the identity of its peer. Instead, each party includes a MAC, keyed with a key derived from g^ab, and applied to its own identity (concatenated with Na and Nb). JFKr inherits the same basic core security as this protocol. In addition, JFKr adds a preliminary cookie mechanism for DoS protection (which results in adding one flow to the protocol and having the *responder* in JFKr play the role of A), and encrypts the last two messages in order to provide identity protection. (Here the identity protection covers both parties, since the identities are sent only in the last two messages.)

The next step in the analysis is to generalize to the case where the private DH exponents are re-used across sessions. This is done by making stronger (but still reasonable) computational intractability assumptions involving both DH problem and the HMAC pseudo random function. We defer details to the full analysis paper.

7. IANA Considerations

IANA is asked to assign a UDP port for JFK at the time that this draft becomes an RFC. Also, the algorithm identifiers will need to be kept in an IANA registry. These two requests will be described in more detail in a future version of this draft.

8. Acknowledgments

We would like to thank Paul Hoffman for suppling us with the draft text in Section 4 (Wire Format), and his constant prodding us in getting this document done. Ran Atkinson, Matt Crawford, and Eric Rescorla provided useful comments, and discussions with Hugo Krawczyk proved very useful. Dan Harkins suggested the inclusion of IPi in the authenticator.

Appendix A. Protocol JFKi

In the following, "I->R" means a message from the Initiator to the Responder and "R->I" means the opposite direction.

```
Message 1, I->R:  Ni, g^i, IDr'

Message 2, R->I:  Ni, Nr, g^r, GRPINFOr, IDr,
                  SIG{r}(g^r, GRPINFOr),
                  HMAC{HKr}(g^r, Nr, Ni, IPi)

Message 3, I->R:  Ni, Nr, g^i, g^r, HMAC{HKr}(g^r, Nr, Ni, IPi),
                  E{Ke}(IDi, sa, SIG{i}(Ni, Nr, g^i, g^r, IDr, sa))

Message 4, R->I:  Ni, Nr,
                  E{Ke}(SIG{r}(Ni, Nr, g^i, g^r, IDi, sa, sa'), sa')
```

The key used to protect Messages (3) and (4), Ke, is computed as

HMAC{g^ir}(Ni, Nr, 1). The session key used by IPsec (or any other application), Kir, is HMAC{g^ir}(Ni, Nr, 0).

If more key material is needed, the same mechanism for "key stretching" as specified in [IKEv2] may be used.

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. This message also contains an indication as to which ID the Initiator would like the Responder to use to authenticate. In contrast to JFKr, IDr' is sent in the clear; however, notice that the responder's ID in Message (2) is also sent in the clear, so there is no loss of privacy in this respect.

Message (2) is more complex. Assuming that the Responder accepts the Diffie-Hellman group in the Initiator's message (rejections are discussed elsewhere in this document), he replies with a signed copy of his own exponential (in the same group), information on what secret key algorithms are acceptable for the next message, a random nonce, his identity (certificates or a bit-string identifying his public key), and an authenticator calculated from a secret, HKr, known to the Responder; the authenticator is computed over the two exponentials and nonces, and the Initiator's network address. 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 authenticator also confirms that the sender of Message (3) uses the same address as in Message (1) --- this can be used to detect and counter a ``cookie jar'' DDoS attack. 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 Ni and Nr. The encryption and authentication use algorithms specified in GRPINFOr. 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 HKr 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). Note: It is important that the responder deals 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, were it 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 Ke, which is derived from Ni, Nr, and g^ir (the result of the Diffie-Hellman computation).

Appendix B. Avoiding Fragmentation-based DoS Attacks

Since Message (3) contains certificates, it may be larger than the MTU and thus require fragmentation (with most common certificate formats, this will be the case only if there are more than 2 certificates included in the payload). Thus, it is possible for an attacker to send partial fragments of a Message (3) packet in an attempt to saturate the Responder's reassembly queue and thus deny service to legitimate Initiators. Similar considerations hold for Message (4).

To avoid this, we propose that the following mechanism be used by the underlying operating system:

- The operating system periodically selects a secret value (16 bits for IPv4, 32 bits for IPv6). This value is also provided to the JFK implementation.

- The operating system also maintains two reassembly queues: one for fragments that contain the correct cookie in the ip_id field (for IPv4 packets) or the ip6f_ident field (for IPv6 packets, in the fragmentation header). Both queues are of fixed size. While the privileged queue may be larger (to accomodate higher traffic volumes), this is not strictly necessary.

- The cookie is encoded in the first 16 (or 32) bits of the Nonce of the appropriate party (Ni or Nr). These should be copied verbatim (i.e., without any byte-swapping etc.) to the appropriate field in the IPv4 header or the IPv6 fragmentation header.

- The cookie is computed by applying a hash function on the peer's IP address and the host secret, then taking the first 16 or 32 bits. Both the operating system and the JFK implementation can perform this computation. The actual algorithm used to compute the cookie is left up to the implementation; a fast keyed hash or equivalent algorithm should be used.

Notice that if multiple retransmissions of the same packet are

performed, all fragments of all instances of the same packet will have the same identification number (whereas they are supposed to have different values). Since, however, no field of Message (3) or (4) changes between retransmissions, this poses no problem even if the sender's network stack fragments the packets in different ways.

Finally, notice that no changes are required on the protocol, and the Initiator's and Responder's operating systems may or may not implement this functionality independently, without affecting interoperability.

References:

[BDJR97] Bellare, Desai, Jokippi, Rogaway, "A Concrete Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", 1997, http://www-cse.ucsd.edu/users/mihir/.

[BR93] Bellare and Rogaway, "Entity authentication and key distribution", Crypto '93. Available at http://www-cse.ucsd.edu/users/mihir.

[CK01] Canetti and Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels", Eurocrypt 01. Available at http://eprint.iacr.org/2001/040.

[CK02] Canetti and Krawczyk, "Security Analysis of IKE's Signature-based Key-Exchange Protocol", manuscript, 2002.

[IKEv2] Harkins, Kaufman, Kent, Kivinen, Perlman, "Proposal for the IKEv2 Protocol", draft-ietf-ipsec-ikev2-01.txt. February 2002. Work in Progress.

[K01] Krawczyk, "The IKE-SIGMA Protocol". Available at http://tiger.technion.ac.il/~hugo/draft-krawczyk-ipsec-ike-sigma-00.txt. Nov 2001. Work in progress.

[RFC2104] Krawczyk, Bellare, Canetti, "HMAC: Keyed-Hashing for Message Authentication. RFC 2104. February 1997.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC2522] Karn, Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, March 1999.

Authors' addresses:

The authors as a group can be reached by email at jfk@crypto.com

William Aiello
AT&T Labs - Research
180 Park Avenue
Florham Park, New Jersey 07932-0971

Email: aiello@research.att.com

Steven M. Bellovin
AT&T Labs - Research
180 Park Avenue
Florham Park, New Jersey 07932-0971

Email: smb@research.att.com

Matt Blaze
AT&T Labs - Research
180 Park Avenue
Florham Park, New Jersey 07932-0971

Email: mab@research.att.com

Ran Canetti
IBM T.J. Watson Research Center
30 Saw Mill Rover Road
Hawthorne, New York 10532
Email: canetti@watson.ibm.com

John Ioannidis
AT&T Labs - Research
180 Park Avenue
Florham Park, New Jersey 07932-0971

Email: ji@research.att.com

Angelos D. Keromytis
Columbia University, CS Department
515 CS Building
1214 Amsterdam Avenue, Mailstop 0401
New York, New York 10027-7003

Phone: +1 212 939 7095
Email: angelos@cs.columbia.edu

Omer Reingold
AT&T Labs - Research
180 Park Avenue
Florham Park, New Jersey 07932-0971

Email: omer@research.att.com

Expiration and File Name

This draft expires in September 2002

Its file name is draft-ietf-ipsec-jfk-04.txt