Derivation of the JFK Protocol

Anupam Datta, John Mitchell and Dusko Pavlovic

Abstract

We introduce a basic framework for deriving security protocols from simple components, like proofs are derived from axioms. As an initial case study, we derive the recently proposed key agreement protocol JFK (Just Fast Keying), starting from the basic Diffie-Hellman exchange, and the generic challenge-response scheme, and refining them by the other required security features, formalized in our derivation system by suitable refinement and transformation rules.

1. Introduction

JFK [1] is a protocol recently proposed to replace IKE [2] as the standard key exchange protocol for the IPSec protocol suite. It has been recognized that IKE suffers from 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. Besides providing a means for authenticated key exchange, JFK has been engineered with the specific design goal of removing these deficiencies.

In this paper, we present a formal analysis of the JFK protocol. The goal has been to verify whether JFK satisfies all its stated design goals. Towards this end, we have developed a "rational reconstruction" of the core JFK protocol. The Diffie-Hellman key exchange protocol [14] and a simplified version of the standard challenge-response authentication protocol [15] constitute the starting point of the reconstruction. We compose these two protocols to obtain the first approximation of an authenticated key exchange protocol. This protocol is then systematically transformed into one that provides DoS protection. We then progressively refine the obtained protocol by adding message components, finally culminating in a protocol that is a close approximation of the actual JFK protocol. At each refinement step, we clearly explain what purpose is served by that message component and/or what attack would arise if it were not executed. We believe that this reconstruction provides a natural way to understand how the message components of the protocol serve to meet the stated design goals: security, identity protection, DoS protection, etc.

The remainder of this paper is structured as follows. Section 2 describes the the design goals of the JFK protocol. Section 3 discusses the protocol itself and the role served by the different message components. In Section 4, we present a systematic reconstruction of the core JFK protocol using compositions, refinements and transformations. Section 5 presents a protocol transformation technique which adds DoS protection to the base protocol. Concluding remarks and directions for future work appear in Section 6.

2. Design Goals

The JFK protocol was designed to meet the following requirements:

- 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 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 the main motivation is to avoid the complexity of IKE. 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.

The Memory-DoS and Computation-DoS properties have become more important in the context of recent Internet denial-of-service attacks. Photuris [21] was the first published key management protocol for which DoS-resistance was a design consideration. 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. The basic JFK protocol provides identity protection for the initiator against active attacks, and no protection for the responder.

The Efficiency property is worth discussing. In many protocols, key setup 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.

Perfect Forward Secrecy (PFS) is treated differently from other protocols. The amount of forward secrecy is treated as an engineering parameter that can be traded off against other necessary functions, such as resistance to denial-of-service attacks. JFK has the concept of a "forward secrecy interval"; associations are protected against compromises that occur outside of that interval.

3. The JFK Protocol

3.1 Notation

The following notation is used in describing the protocol.

$A \rightarrow B$	Message from A to B
$E_K(M)$	Encryption of M with symmetric key K
$HMAC_K(M)$	Keyed hash (HMAC) of M using key K
$SIG_x(M)$	Signature of x on message M
g^x	Diffie-Hellman (DH) exponentials
N_x	Random nonce
sa	Initiator desired Security Association
sa'	Responder's IPSec SPI
HKr	Responder's transient private hash key
K_e	Encryption key: $HMAC_{g^{ir}}(N_i, N_r, 1)$
K_{ir}	Session key: $HMAC_{g^{ir}}(N_i, N_r, 0)$
ID_x	Public key certificate of x
$GRPINFO_r$	Responder supported DH groups

3.2 Protocol Description

The JFK protocol is shown in Figure 1. 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. This message also contains an indication as to which ID the Initiator would like the Responder to use to authenticate.

Message 2 is more complex. Assuming that the Responder accepts the Diffie-Hellman group in the Initiator's message, 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. This is meant to ensure that the Responder is not open to denial-service attacks.

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

$$I \rightarrow R : N_i, g^i, ID_{r'}$$

$$R \rightarrow I : N_i, N_r, g^r, GRPINFO_r, ID_r, SIG_r\ (g^r, GRPINFO_r), HMAC_{HKr}(N_i, N_r, g^i, g^r, IP_i)$$

$$I \rightarrow R : N_i, N_r, g^i, g^r, HMAC_{HKr}\ (N_i, N_r, g^i, g^r, IP_i), E_{K_e}\ (ID_i, sa, SIG_i(N_i, N_r, g^i, g^r, ID_r, sa))$$

$$R \rightarrow I : E_{K_e}(SIG_r(N_i, N_r, g^i, g^r, ID_i, sa, sa'), sa')$$

Figure 1. JFK Protocol

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 $GRPINFO_r$. 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 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 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 K_e, which is derived from N_i, N_r, and g^{ir} (the result of the Diffie-Hellman computation).

4. Reconstructing JFK

In this section, we reconstruct JFK by applying refinements and transformations to a protocol obtained by composing the Diffie-Hellman key exchange protocol with the standard challenge-response based authentication protocol. A *refinement* replaces every instance of a message component used in the protocol by another. In this sense, it is a local operation, somewhat like the find-and-replace operation in a text editor which substitutes every instance of a given pattern by another. On the other hand, a *transformation* involves a series of steps and operates on the protocol as a whole. In particular, we describe a transformation that adds "cookies" to the base protocol to make it resistant to DoS attacks. This reconstruction provides a natural way to understand how the message components of the protocol serve to meet the stated design goals: security, identity protection, DoS protection, etc.

1. Components:

   (a) Diffie-Hellman key exchange: The basic Diffie-Hellman protocol [14] is shown below. It provides a way for two parties to set up a shared key (g^{ir}) which a passive eavesdropper cannot recover. The security of the key depends on the computational hardness of the discrete logarithm problem.

   $$I \rightarrow R : g^i$$

   $$R \rightarrow I : g^r$$

   (b) Challenge-response: The signature-based challenge-response protocol shown below is a standard mechanism for providing mutual authentication (see Section 10.3.3 of [15]).

   $$I \rightarrow R : m$$

   $$R \rightarrow I : n, SIG_r\ (n, m)$$

   $$I \rightarrow R : SIG_i(m, n)$$

2. Transformations:

   (a) The "cookie transformation" is discussed in detail in the next section. At the cost of adding an extra message to the base protocol, it guarantees that the Responder does not have to create state or perform expensive computation before a round-trip communication is established with the Initiator. This helps protect the Responder against both Computation-DoS and Memory-DoS attacks.

   (b) The second transformation rule allows a field t of message msg_i to be moved to an earlier message msg_j ($j < i$) with the same sender and receiver, provided t does not contain any data freshly generated between the two messages. Additional side conditions could be imposed to ensure secrecy requirements on t.

3. Refinements:

   In what follows, we use $a \Longrightarrow b$ to denote that b should replace every instance of a in the protocol once that refinement is applied.

   (a) $SIG_X(m) \Longrightarrow E(SIG_X(m))$, where E denotes encryption under some shared key. This refinement is necessary for identity protection. Since everything signed is public except the sa, and that is often guessable, an attacker could verify guesses at identities if the signature was not encrypted. In JFK, a shared key derived from the Diffie-Hellman secret and the two nonces is used for encryption.

   (b) $SIG_X(m) \Longrightarrow SIG_X(m), ID_X$, where ID_X denotes the public-key certificate of X. Since the other party may not possess the signature-verification key, it is necessary to include the certificate along with the signature.

   (c) $SIG_X(m) \Longrightarrow SIG_X(m, Y)$, where Y is the peer's identity. A side condition here is that X possesses the requisite identifying information for Y, e.g., $Y's$ public key certificate, before the protocol is executed. This condition can be removed if X receives $Y's$ identity in an earlier message of the protocol. In public-key based challenge-response protocols, the authenticator should identify both the sender and the intended recipient. Otherwise, the protocol is susceptible to a person-in-the-middle-attack. Here, the signature identifies the sender and the identity inside the signature identifies the intended recipient. In an encryption-based challenge-response protocol (e.g., Needham-Schroeder [23]), since the public encryption key identifies the intended recipient, the sender's identity needs to be included inside the encryption. The original protocol did not do so and the bug was discovered nearly twenty years later by Lowe [24].

   (d) $g^x \Longrightarrow g^x, N_x$, where N_x is a random nonce generated by the same entity which generated g^x. The nonce is a fresh random value and allows Diffie-Hellman exponentials to be reused across multiple protocol runs.

   Note that refinements (b) and (c) are included as part of the base challenge-response protocol in [15] (Section 10.3.3). We present them separately here in order to clearly explain their significance.

We will now present a step-by-step reconstruction of the core JFK protocol using these components, transformations and refinements.

- Step 1: Compose components 1(a) and 1(b) above by substituting g^i for m and g^r for n in 1(b). The resulting protocol is shown below.

$$I \rightarrow R : g^i$$
$$R \rightarrow I : g^r, SIG_r\ (g^r, g^i)$$
$$I \rightarrow R : SIG_i(g^i, g^r)$$

- Step 2: Apply the "cookie transformation" to the protocol obtained from Step 1.

$$I \rightarrow R : g^i$$
$$R \rightarrow I : g^r, HMAC_{HKr}(g^i, g^r)$$
$$I \rightarrow R : g^i, g^r, HMAC_{HKr}(g^i, g^r), SIG_i\ (g^i, g^r)$$
$$R \rightarrow I : SIG_r\ (g^r, g^i)$$

- Step 3: Apply refinement 3(a) to the protocol obtained from Step 2 using K_e as the encryption key.

$$I \rightarrow R : g^i$$
$$R \rightarrow I : g^r, HMAC_{HKr}(g^i, g^r)$$
$$I \rightarrow R : g^i, g^r, HMAC_{HKr}(g^i, g^r)$$
$$E_{K_e}(\ SIG_i\ (g^i, g^r))$$
$$R \rightarrow I : E_{K_e}(\ SIG_r\ (g^r, g^i))$$

- Step 4: Apply refinement 3(b) to the protocol obtained from Step 3.

$$I \rightarrow R : g^i$$
$$R \rightarrow I : g^r, HMAC_{HKr}(g^i, g^r)$$
$$I \rightarrow R : g^i, g^r, HMAC_{HKr}(g^i, g^r)$$
$$E_{K_e}(\ SIG_i\ (g^i, g^r), ID_i)$$
$$R \rightarrow I : E_{K_e}(\ SIG_r\ (g^r, g^i), ID_r)$$

- Step 5: Apply transformation 2(b) to move the field ID_r from message 4 to message 2.

$$I \rightarrow R : g^i$$

$$R \rightarrow I : g^r, HMAC_{HKr}(g^i, g^r), ID_r$$

$$I \rightarrow R : g^i, g^r, HMAC_{HKr}(g^i, g^r)$$

$$E_{K_e}(\ SIG_i\ (g^i, g^r), ID_i)$$

$$R \rightarrow I : E_{K_e}(\ SIG_r\ (g^r, g^i))$$

- Step 6: Apply refinement 3(c) to messages 3 and 4 of the protocol obtained from Step 4. Note that the side condition for applying this rule to message 3, whereby I needs to know peer's identity ID_r before she signs it, is satisfied after step 5. The side condition for applying it to message 4 is also satisfied, because ID_i is introduced in message 4.

$$I \rightarrow R : g^i$$

$$R \rightarrow I : g^r, HMAC_{HKr}(g^i, g^r), ID_r$$

$$I \rightarrow R : g^i, g^r, HMAC_{HKr}(g^i, g^r)$$

$$E_{K_e}(\ SIG_i\ (g^i, g^r, ID_r), ID_i)$$

$$R \rightarrow I : E_{K_e}(\ SIG_r\ (g^r, g^i, ID_i))$$

- Step 7: Apply refinement 3(d) to the protocol obtained from Step 6.

$$I \rightarrow R : g^i, N_i$$

$$R \rightarrow I : g^r, N_r, HMAC_{HKr}(g^i, N_i, g^r, N_r), ID_r$$

$$I \rightarrow R : g^i, N_i, g^r, N_r, HMAC_{HKr}(g^i, N_i, g^r, N_r)$$

$$E_{K_e}(\ SIG_i\ (g^i, N_i, g^r, N_r, ID_r), ID_i)$$

$$R \rightarrow I : E_{K_e}(\ SIG_r\ (g^r, N_r, g^i, N_i, ID_i))$$

The protocol obtained from Step 7 represents the core JFK protocol. For ease of exposition, we have ignored some message components, viz. $ID_{r'}$, $GRPINFO_r$, sa, sa', IP_i. This version has the following desirable properties: security, identity protection (for initiator against passive attackers only), computation DoS protection, memory DoS protection, and "almost" perfect forward secrecy. Since $SIG_r(g^r)$ is not included in message 2, this protocol does not provide identity protection for the initiator against active attackers. Extending the composition-refinement based framework to obtain a reconstruction of the complete JFK protocol would be an interesting challenge.

5. The "Cookie" Transformation

In this section, we describe the "cookie transformation": a mechanism that makes a protocol resistant to DoS attacks. The goal is to describe a transformation that systematically transforms the protocol in Step 1 of the previous section into the protocol obtained in Step 2. We will do this in 3 stages. First we describe two transformations which separately provide protection against computation DoS and memory DoS attacks. We then compose these to obtain the desired transformation.

5.1 Protocol Description Notation

We use an extension of the strand space notation [25], inspired by [4] to explicitly denote what constitutes a computation DoS/ memory DoS attack. Here, we explain the notation with a simple example. Consider the following 3-step protocol from Step 1 of the previous section:

$$I \rightarrow R : g^i$$

$$R \rightarrow I : g^r, SIG_r\ (g^r, g^i)$$

$$I \rightarrow R : SIG_i(g^i, g^r)$$

The representation of this protocol in the extended strand space notation is shown in Figure 2. The horizontal arrows denote messages exchanged during the protocol. The views of both the participants are explicitly shown. For example, in the first message, since I generates g^i, it is shown as such in I's view. But as far as R is concerned, it is just a random number. R has no way of verifying that it is of the form g^i since that would involve computing discrete logarithm. So, it is denoted by x. A similar reasoning applies to g^i and u in the second message. Since I knows that the second component of message 2 is R's signature and can verify it, it is explicitly shown as a signature. Similarly, signatures are shown as such in message 3.

Vertical arrows are used to denote state changes upon sending and receiving messages. We distinguish between dashed vertical arrows (which denote internal computation involved in processing a received message) and solid vertical arrows (which denote waiting-for-message phases). Beside a solid vertical arrow, we specify the parameters saved in the local state at that point. For example, after sending message 2, R saves x, r in its local state till it receives message 3. The dashed vertical arrows denote possible sites for Computational DoS attacks while the solid vertical arrows denote possible sites for Memory DoS attacks. We label a single vertical arrow E^* if it requires the participant to perform computationally expensive operations.

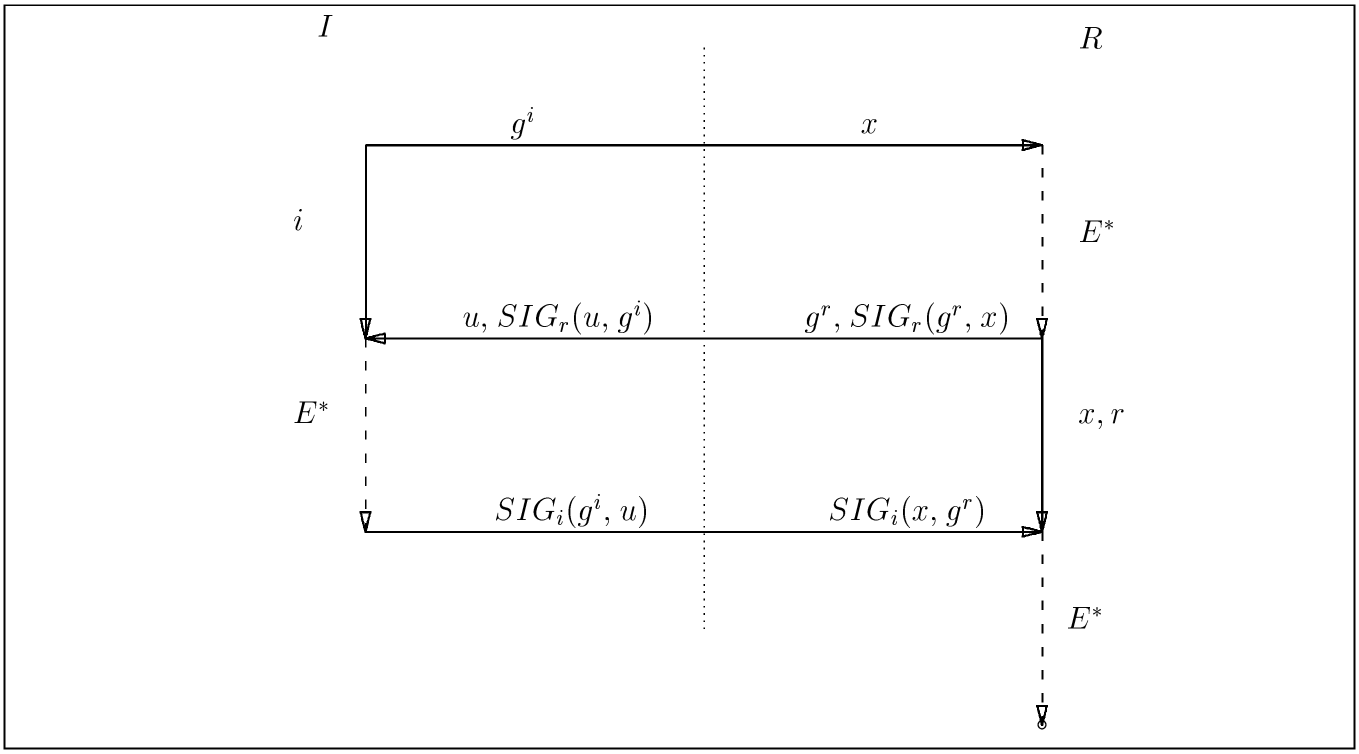

Figure 2. Example

5.2 The Transformation

1. CDoS protection:

   The idea is to establish round-trip communication with I before performing expensive operations. R sends some fresh data to I which can be generated without performing expensive operations. I has to return that piece of data back to complete the round-trip. Freshness is required for replay protection. In the notation defined in the previous section, the first dashed vertical arrow on R's side should not contain E^*.

   The transformation is shown in Figure 3. Here n is the fresh data. m denotes the piece of information using which n can be reconstructed from $msg1$. Computation of n should be inexpensive. $msg2(E-)$ and $msg2(E+)$ denote respectively the components of $msg2$ that do not require/require expensive operations; here, $msg2(E-) = g^r$ (since Diffie-Hellman exponentials are reused) and $msg2(E+) = SIG_r(g^r, x)$.

2. MDoS protection:

   The idea is that R should not save state before round-trip communication is established with I. Instead she should send out an unforgeable "token" that captures the state and which can be used later to reconstruct the state. I should send back this token in the next message. For example, a keyed hash of the DH exponentials serves this purpose in JFK. The transformation is shown in Figure 4. Here C corresponds to the tuple (g^r, x). The goal here is to ensure that the first solid vertical arrow on R's side should not contain state information (beyond HKr which is reused across multiple sessions and need not be counted as part of local session state).

3. CDoS and MDoS protection:

   We compose the transformations of Figure 3 and Figure 4 by applying the following rules:

   (a) Set $n = C, HMAC_{HKr}(C)$ and $m = HKr$.

   (b) Take union of components of corresponding messages.

   (c) Preserve computation constraints on dashed vertical arrows.

   (d) Preserve message ordering for each participant.

   (e) Do not repeat message components which are already known.

The resulting 5-message protocol is shown in Figure 5. Components in square brackets denote parameters that are reused across multiple sessions and hence need not be counted as part of local session state. Note that this

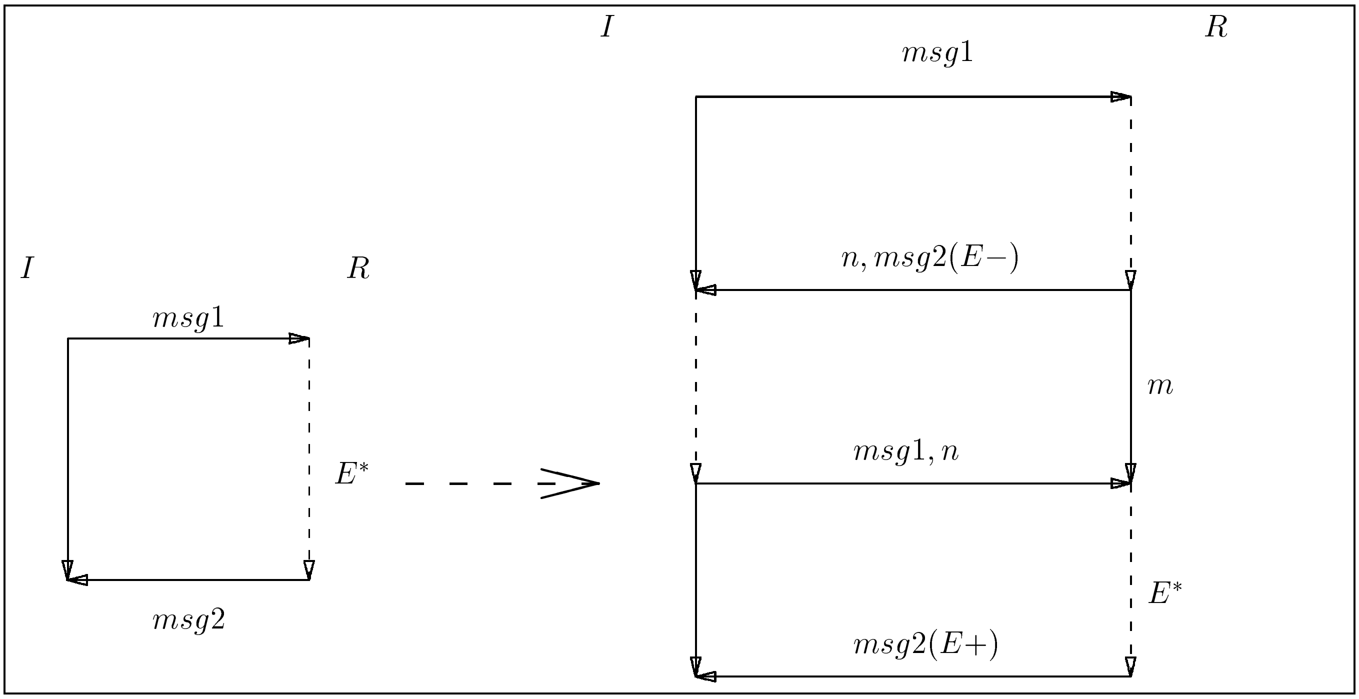

Figure 3. CDoS Protection Transformation

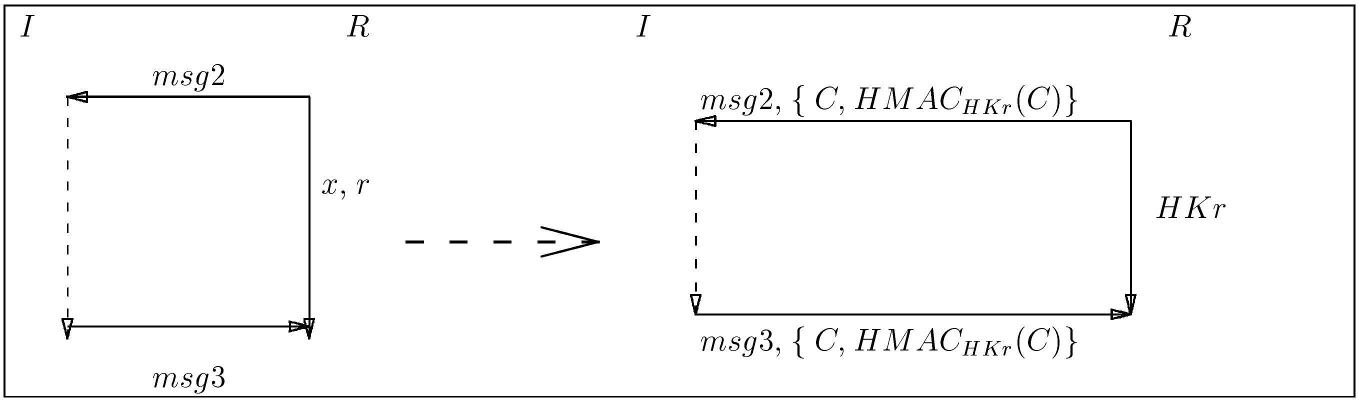

Figure 4. MDoS Protection Transformation

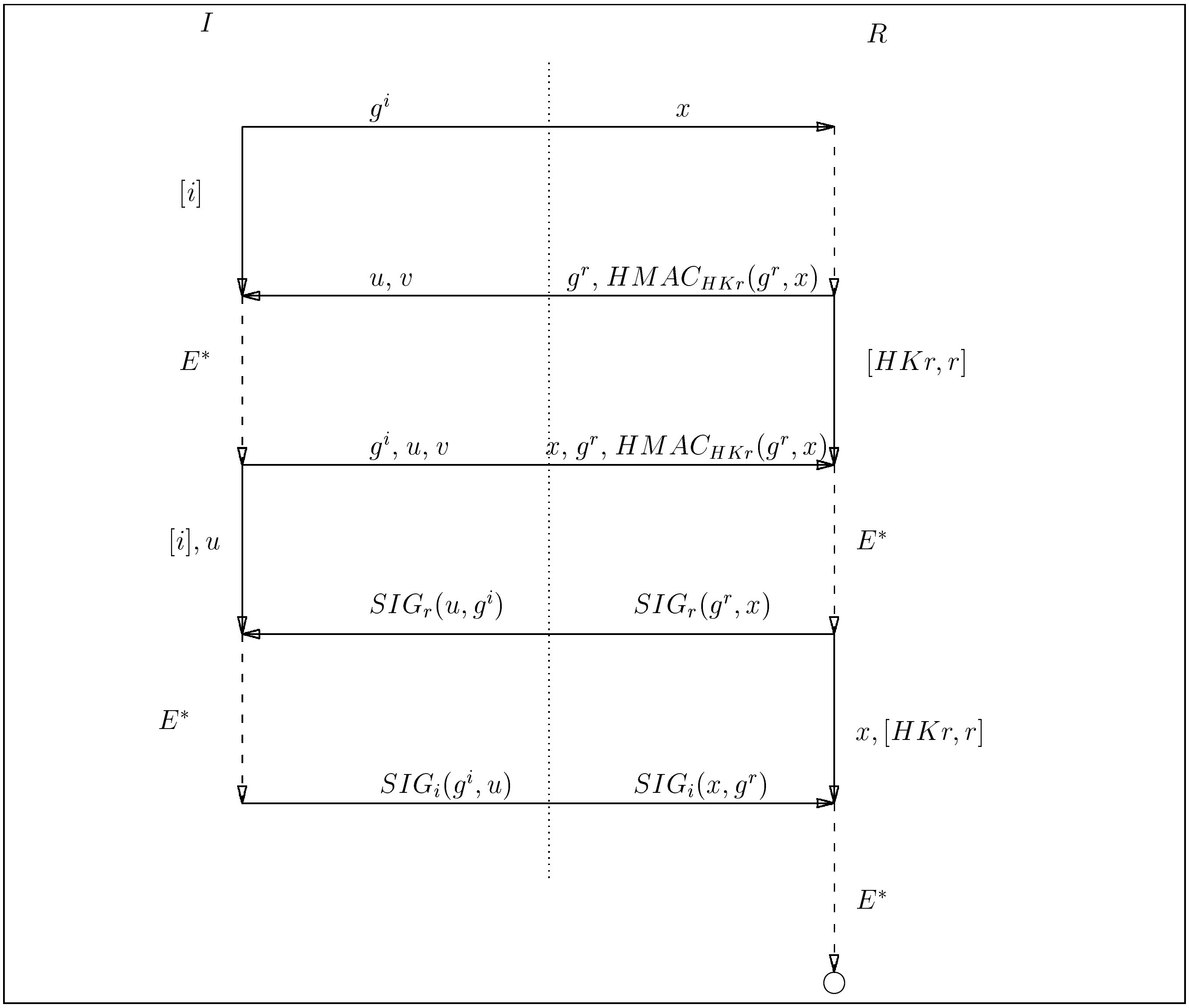

Figure 5. DoS Protection Transformation

protocol is resistant to both blind computation DoS and memory DoS attacks. It is possible to reduce the number of messages by applying the following rule. Start from the last message. If all components are computable when the previous message was sent by this participant, then move these components to the previous message, eliminate this message and iterate. (It is necessary to check that computation constraints are not violated before performing this step.) This is essentially the same rule as transformation 2(b) in the previous section. We can apply this rule to the protocol of Figure 5 to eliminate message 5 and move the signature to message 3. Further iteration cannot proceed because of the computation constraint on the first vertical arrow of R. The resulting protocol is shown in Figure 6. It is identical to the one obtained from Step 2 in the previous section.

6. Conclusions and Future Work

We have presented a formal analysis of the JFK protocol. The goal has been to verify whether JFK satisfies all its stated design goals. Towards this end, we have developed a "rational reconstruction" of the core JFK protocol. Starting from the Diffie-Hellman key exchange protocol [14] and a simplified version of the standard challenge-response authentication protocol [15], we systematically reconstructed a close approximation of the JFK protocol using a series of protocol compositions, transformations and refinements. At each refinement step, we have clearly explained what purpose is served by that message component and/or what attack would arise if it were not executed. We believe that this reconstruction provides a natural way to understand how the message components of the protocol serve to meet the stated design goals. Also, the general problem of composing security protocols in such a way that the security properties of both the constituent protocols are retained, has been recognized as quite a difficult one. The method for composing Diffie-Hellman key exchange with challenge-response as well as the composition technique for combining CDoS protection with MDoS protection, although special cases, do provide some indication that a general framework for composing security protocols may be achievable.

References

[1] W. Aiello, S.M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A.D. Keromytis, O. Reingold. Just Fast Keying (JFK). *Internet Draft*, November 2001.

[2] D. Harkins, D. Carrel. The Internet Key Exchange (IKE). *RFC 2409*, November 1998.

[3] D. Dill. The Murφ Verification System. *In Proc. 8th International Conference on Computer Aided Verification*, pages 390-393, 1996.

[4] N. Durgin, J. Mitchell and D. Pavlovic. A Compositional Logic for Protocol Correctness. *In Proc. of CSFW 2001*, pages 241-255, IEEE 2001.

[5] R. Kemmerer, C. Meadows, J. Millen. Three Systems for Cryptographic Protocol Analysis. *In Journal of Cryptography*, 7(2):79-130, 1994.

[6] A.W.Roscoe. Modelling and Verifying Key Exchange Protocols using CSP and FDR. *In Proc. 8th Computer Security Foundations Workshop*, pages 98-107, 1995.

[7] C. Meadows. The NRL Protocol Analyzer: An Overview. *In Journal of Logic Programming*, 26(2):113-131, 1996.

[8] D. Bolignano. Towards a mechanization of cryptographic protocol verification. *In Proc. 9th International Conference on Computer Aided Verification*, 131-142, 1997.

[9] L. Paulson. The inductive approach to verifying cryptographic protocols. *In Journal of Computer Security*, 6:85-128, 1998. 131-142, 1997.

[10] J.C. Mitchell, M. Mitchell, U. Stern . Automated Analysis of Cryptographic Protocols Using Murφ. *In Proc. IEEE Symposium on Security and Privacy*, pages 141-153, 1997.

[11] J.C. Mitchell, V. Shmatikov, U. Stern . Finite-State Analysis of SSL 3.0. *In Proc. 7th USENIX Security Symposium*, pages 201-216, 1998.

[12] V. Shmatikov, J.C. Mitchell. Analysis of a Fair Exchange Protocol. *In Proc. 7th Annual Symposium on Network and Distributed System Security*, pages 119-128, 2000.

[13] V. Shmatikov, J.C. Mitchell. Analysis of a Abuse-Free Contract Signing Protocol. *In Proc. Financial Cryptography*, 2000.

[14] W. Diffie, M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644-654, 1976.

[15] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone. Handbook of Applied Cryptography. *CRC Press*, 1996.

[16] W. Diffie, P. C. Van Oorschot, M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2:107-125, 1992.

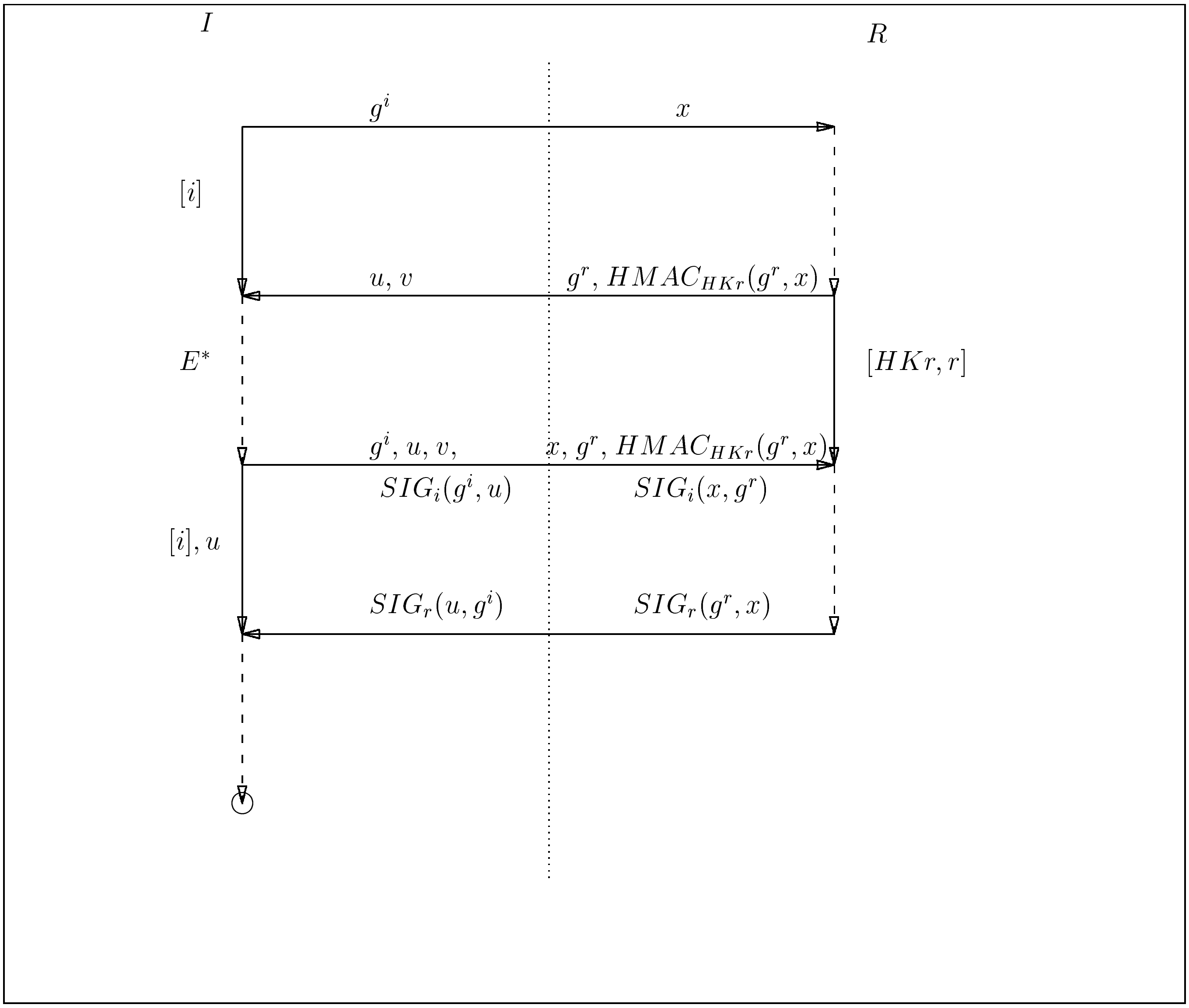

Figure 6. Optimized Transformed Protocol

[17] H. Krawczyk, M. Bellare, R. Canetti. HMAC: Keyed-Hashing for Message Authentication. *RFC 2104*, February 1997.

[18] D. Dill, S. Park, A. G. Nowatzyk. Formal Specification of Abstract Memory Models. *In Symposium on Research on Integrated Systems*, pages 38-52, 1993.

[19] U. Stern, D. Dill. Automatic Verification of the SCI Cache Coherence Protocol. *In Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 21-34, 1995.

[20] V. Shmatikov, U. Stern . Efficient Finite-State Analysis for Large Security Protocols. *In Proc. 11th IEEE Computer Security Foundations Workshop*, pages 106-115, 1998.

[21] P. Karn, W. Simpson. Photuris: Extended Schemes and Attributes. *RFC 2523*, March 1999.

[22] H. Krawczyk. The IKE-SIGMA Protocol. *Internet Draft*, November 2001.

[23] R. Needham, M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12): 993-9, 1978.

[24] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using CSP and FDR. *In Proc. of 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1996.

[25] F. J. T. Fabrega, J. C. Herzog, J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*,7(2/3):191–230, 1999.