STUDIES IN
AUTHENTICATED KEY ESTABLISHMENT PROTOCOLS

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE WITH DISTINCTION IN RESEARCH

By
Anupam Datta
May 17, 2002

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science with Distinction in Research.

John C. Mitchell
(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science with Distinction in Research.

David L. Dill

Acknowledgements

I would like to thank my advisor, John Mitchell, for establishing the direction of this research and also for his time and patience in explaining the major research problems in this area. This work was done in collaboration with Dusko Pavlovic, Frederic Muller and Ante Derek. We had a number of fruitful discussions and I would like to thank them for their comments and suggestions. My gratitude also goes to Anupam Basu, my undergraduate advisor, who initiated me to research, and who greatly helped me to come to Stanford. Finally, I would like to thank my family for their continuous support and encouragement, and for teaching me how to think (mostly) rationally.

Anupam Datta
Stanford, California
May 17, 2002

Contents

Chapter 1

Introduction

A *cryptographic protocol* is a multi-party algorithm, defined by a sequence of steps precisely specifying the actions required of two or more parties in order to achieve a specified objective. Based on their objectives, the following types of protocols may be distinguished: an *authentication protocol* provides to one party some degree of assurance regarding the identity of another with which it is purportedly communicating; a *key establishment protocol* is used to establish a shared secret between two or more parties; and finally an *authenticated key establishment protocol* serves both these purposes, i.e., it can be used to establish a shared secret with a party whose identity has been or can be corroborated. Since these protocols are executed over a network, it is important to understand the underlying assumptions about the network, in order to clarify the threats they may be subject to. Typically, the network is assumed to be hostile, in that it may contain intruders who can read, modify, and delete traffic, and who may have control of one or more parties involved in the protocol. Because of this, such protocols are often subject to non-intuitive attacks which are not easily apparent even to a careful inspector.

Since formal methods allow one both to do a thorough analysis of the different paths which an intruder can take, and to specify precisely the environmental assumptions that have been made, they have proved quite useful in analyzing the security of cryptographic protocols. The first major work in this area was done by Dolev and Yao [1]. This work was significant in that it was the first to develop a formal model of an environment in which multiple executions of the protocol can be running concurrently, in which cryptographic algorithms behave like black boxes which obey a limited set of algebraic properties (e.g. the encryption and decryption operations cancel each other out), and which includes an intruder who can read, alter, and destroy traffic, and may also control some legitimate members of the system. Most later work on the formal analysis of cryptographic protocols is based on this model or some variant of it.

Recent research has focussed on state exploration tools and theorem proving techniques based on the Dolev-Yao model, much of it sparked by Lowe's demonstration that it was possible to use a general-purpose model-checker, FDR, to find a man-in-the-middle attack on the Needham-Schroeder public key protocol [2]. Work since then has progressed in applying both model-checkers [3, 4] and theorem-provers [5, 6] to the problem, as well as in the design of special-purpose model-checkers [7, 8, 9] and the use of specialized tools originally intended for somewhat different applications [10].

1.1. Synopsis and Outline of the Thesis

In this thesis, we present a formal analysis of two independent authenticated key establishment protocols. The first protocol provides authentication for Mobile IPv6 connections [12] while the second - Just Fast Keying (JFK) [36] - has been recently proposed to replace IKE [37] as the standard protocol for the IPSec protocol suite. We carried out a formally analysis of both these protocols using the model-checking tool Murφ [17]. In previous work, Murφ has been used to analyze the security properties of several protocols. Chapter 2 describes the Murφ verification system and outlines the methodology for using it to analyze security protocols. This analysis is based on the Dolev-Yao model.

In Chapter 3, we present a protocol for authenticating Mobile IPv6 connections, addressing requirements established in the relevant Internet Draft [11]. The protocol imposes minimal computational requirements on mobile nodes, uses as few messages as possible, and may be adapted to resist denial of service attacks. Our protocol has two parts, an initialization phase and an update phase. The initialization phase can take advantage of local public-key infrastructure, a bidirectional chain of trust, or operate without prior authenticating information. Each execution of the update phase uses the shared secret established in the previous phase to maintain security of the mobile connection. We believe that our protocol addresses the main issues in Mobile IPv6 authentication.

Chapter 4 presents 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 [27] and a simplified version of the standard challenge-response authentication protocol [28] 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. 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 we present for combining Computational DoS protection with Memory DoS protection, although special cases, do provide some indication that a general framework for composing security protocols may be achievable.

Concluding remarks and directions for future work in the analysis of these two protocols appear separately in the last section of Chapter 3 and of Chapter 4.

Chapter 2

Finite-state Analysis using Murφ

We used Murφ, a finite-state analysis tool, to carry out a formal analysis of the authentication protocol for Mobile IPv6 as well as the JFK protocol. In previous work, Murφ has been successfully used to analyze several security protocols, e.g., Needham-Schroeder, SSL, and contract-signing protocols (see [3, 22, 23, 24]). In this chapter, we briefly outline the general methodology used in the Murφ analysis of security protocols. A detailed description of the methodology can be found in [3].

2.1. The Murφ Verification System

Murφ [17] is a finite-state verification tool that has been successfully applied to multiprocessor cache coherence protocols and multiprocessor memory models [31, 32]. The purpose of finite-state analysis (also called *model checking*) is to exhaustively search all possible execution sequences. While this process often reveals errors, failure to find errors does not imply that the protocol is completely correct, because the Murφ model may simplify certain details and is inherently limited to configurations involving a small number of protocol participants.

To use Murφ for verification, one has to model the protocol in the Murφ language and augment this model with a specification of the desired properties. The Murφ system automatically checks, by explicit state enumeration, if all reachable states of the model satisfy the given specification. For the state enumeration, either breadth-first or depth-first search can be selected. Reached states are stored in a hash table to avoid redundant work when a state is revisited. The memory available for this hash table typically determines the largest tractable problem.

The Murφ language is a simple high-level language for describing non-deterministic finite-state machines. Many features of the language are familiar from conventional programming languages. The main features not found in typical high-level programming languages are described in the following paragraphs.

The *state* of the model consists of the values of all global variables. In the *startstate* statement, initial values are assigned to global variables. The transition from one state to another is performed by *rules*. Each rule has a Boolean condition and an action, which is a program segment that is executed atomically. The action may be executed if the condition is true (i.e., the rule is enabled) and typically changes global variables, yielding a new state. Most Murφ models are nondeterministic since states typically allow execution of more than one rule. For example, in the model of our authentication protocol, the intruder (which is part of the model) usually has the choice of several messages to replay.

Murφ has no explicit notion of *processes*. Nevertheless a process can be implicitly modeled by a set of related rules. The *parallel composition* of two processes is simply done by taking the union of the rules of the two processes. Each process can take any number of steps (actions) between the steps of the other. The resulting model is that of *asynchronous, interleaving concurrency*.

The Murφ language supports *scalable* models. In a scalable model, one is able to change the size of the model by simply changing constant declarations. When developing protocols, one typically starts with a small protocol configuration. Once this configuration is correct, one gradually increases the protocol size to the largest value that still allows verification to complete. In many cases, an error in the general (possibly infinite state) protocol will also show up in a down-scaled (finite state) version of the protocol. Murφ can only guarantee correctness of the down-scaled version of the protocol, but not that of the general protocol. For example, in modelling our authentication protocol, the numbers of mobile nodes and home agents were scalable and defined by constants.

The desired properties of a protocol can be specified in Murφ by *invariants*, which are boolean conditions that have to be true in every reachable state. If a state is reached in which some invariant is violated, Murφ prints an error trace - a sequence of states from the start state to the state exhibiting the problem.

2.2. The Methodology

We analyzed protocols using the following sequence of steps:

1. *Formulate the protocol.* This generally involves simplifying the protocol by identifying the key steps and primitives. The Murφ formulation of a protocol, however, is more detailed than the high-level descriptions often seen in the literature, since one has to decide exactly which messages will be accepted by each participant in the protocol. Since Murφ communication is based on shared variables, it is also necessary to define an explicit message format, as a Murφ type.

2. *Add an adversary to the system.* We assume that the adversary (or intruder) can masquerade as an honest participant in the system, capable of initiating communication with a truly honest participant, for example. We also assume that the network is under the control of the adversary and allow the adversary the following actions:

   - overhear every message, remember all parts of each message, and decrypt ciphertext when it has the key;
   - intercept (delete) messages;
   - generate messages using any combination of initial knowledge about the system and parts of overheard messages.

   Although it is simplest to formulate an adversary that nondeterministically chooses between all possible actions at every step of the protocol, it is more efficient to reduce the choices to those that actually have a chance of affecting other participants.

3. *State the desired correctness conditions.* In authenticated key exchange protocols, a typical correctness condition would be that the intruder does not learn any secret information. Furthermore, whenever a participant A completes a session with participant B, B should also have completed the same session with A or should be waiting for the last message from A.

4. *Run the protocol* for some specific choice of system size parameters. For example, we have been able to run our Mobile IPv6 authentication protocol with upto 3 mobile nodes and 3 home agents, where each mobile node can execute 2 sessions in parallel.

2.3. The Intruder Model

The Murφ intruder model is limited in its capabilities and does not have all the power that a real-life intruder may have. In particular:

- *No cryptanalysis.* Our intruder ignores both computational and number-theoretic properties of cryptographic functions. As a result, it cannot perform any cryptanalysis whatsoever. If it has the proper key, it can read an encrypted message or (forge a signature). Otherwise, the only action it can perform is to store the message for a later replay.

- *No probabilities.* Murφ has no notion of probability. Therefore, we do not model "propagation" of attack probabilities through our finite-state system (e.g., how the probabilities of breaking the encryption, forging the signature, etc. accumulate as the protocol progresses). We also ignore, e.g., that the intruder may learn some probabilistic information about the participants' keys by observing multiple runs of the protocol.

- *No partial information.* Keys, signatures, etc. are treated as atomic entities in our model. Our intruder cannot break such data into separate bits. It also cannot perform an attack that results in the partial recovery of a secret (e.g., half of the secret bits).

In spite of the above limitations, previous studies have shown that Murφ is a useful tool for analyzing security protocols. It considers the protocol at a high level and helps discover a certain class of bugs that do not involve attacks on cryptographic functions employed in the protocol. In fact, Murφ is quite useful in discovering authentication bugs since properties like key ownership, source of messages, etc. are easily captured in logical statements.

Chapter 3

Authentication for Mobile IPv6

3.1. Introduction

In Mobile IPv6 [12], a mobile node has two associated IP addresses. A mobile node is always identified by its *home address.* While operating away from its home, a mobile node is also associated with a *care-of address,* which provides information about its current location on the internet. The care-of address is registered with the home agent, which transparently routes IPv6 packets sent to the home address to the care-of address. To reduce routing distance and relieve the load on home agents, a mobile node may also inform other IPv6 nodes about its current care-of address by sending a *binding update.* The need for authenticating binding updates has been recognized [12, 11]. However, previous authentication proposals either fall short of meeting the authentication requirements [13, 14] or require the mobile nodes to perform expensive public key operations.

In this chapter, we present a protocol for authenticating binding updates. The protocol has two phases: (a) a once-per-connection *initialization phase* in which two mobile nodes use any available chain of trust through their home agents to confirm a shared secret; (b) an *update phase* that is executed every time an authenticated binding update needs to be sent. The protocol provides sender authentication, data integrity, and replay protection, without requiring mobile nodes to perform public key operations. Using a forward message from the mobile node to the corresponding node, and a reply through the home agents only once in the initialization phase, the protocol minimizes the number of messages exchanged between participating entities. While we assume that each mobile node shares a secret key with its home agent, the protocol does not require home agents to maintain state during the execution of the protocol, avoiding memory denial-of-service attacks and other resource consumption problems.

The most difficult adoption issue for any form of authenticated Mobile IP is reliance on authentication infrastructure. Our goal is to make the best use of whatever infrastructure is available, in no case offering less assurance than the link between the two home agents, which in some cases may be wired, unauthenticated IP. As an expository convenience, we present our protocol under the assumption that each home agent has a public-private key pair, with each public key known to the other. We then consider three other scenarios and present variations of the initialization phase that are appropriate to each scenario. The update phase of our protocol remains the same in each case, since the basis for authenticated update is the shared private information established by the initialization phase. While there have been concrete proposals for maintaining a global trust infrastructure, e.g., by extending the Domain Name System (see [15, 16]), we believe our protocol offers significant advantages over previous proposals in the absence of public-key infrastructure. Perhaps the most likely configuration in the near term is an SSH-like scheme, using unauthenticated key exchange between home agents the first time these home agents participate in Mobile IP, and caching the shared keys for future use by any mobile nodes using the same pair of home agents. While this configuration is susceptible to a person-in-the-middle attack, an attack is only possible once for each pair of home agents (up to the capacity of the key cache).

We have formally verified the correctness of our protocol using the finite state analysis tool Murφ [17]. While formal methods have been successfully used to analyze key exchange and authentication protocols [18, 19, 20, 21, 3, 6], this is the first Diffie-Hellman based key exchange protocol that has been analyzed with Murφ. In addition, previously analyzed protocols such as SSL [22] and contract signing [23, 24] involve only two parties while Mobile IP has four (two mobile nodes and their associated home agents). Therefore, the analysis of Mobile IP requires several new concepts and modelling techniques.

The remainder of this chapter is structured as follows. Section 3.2 describes the requirements for security in Mobile IPv6. Section 3.3 briefly discusses the previous proposals for authenticating binding updates. In Section 3.4, we present our basic protocol. Section 3.5 presents our modelling assumptions and analysis results. In Section 3.6, we discuss extensions to prevent denial of service attacks. Section 3.7 describes how our base protocol can be adapted to work in a AAA infrastructure [25, 26] and weaker trust infrastructures. Concluding remarks appear in Section 3.8.

3.2. Security Requirements for Binding Updates

While the binding update feature of Mobile IPv6 makes end-to-end routes shorter, the ability to change routes on the fly introduces a number of security concerns. A Mobile IPv6 binding update specifies an association between the home address of a mobile node and its care-of address, along with the remaining lifetime of that association. Upon receiving a binding update from a mobile node (MN), a correspondent node (CN) creates a binding cache entry and stores this association. Subsequently, the CN will send all packets destined for the MN to its care-of address instead of its home address. Threats and security requirements for binding update are described at length in [11]. Here, we discuss the most pressing security issues.

In the absence of a pre-established security association between MN-CN pairs, two major security threats arise:

- An attacker may tamper with the binding cache entries since binding updates will not be authenticated. In fact, an active attacker can launch a person-in-the-middle attack, becoming the default router to the MN and from the MN to the CN.
- An attacker can launch denial-of-service attacks on MN's, CN's and Home Agents (HA's). This could be done, for example, by flooding IPv6 nodes with fake binding updates.

Our main goal is to institute a mechanism for setting up security associations between MN-CN pairs that will allow binding updates to be authenticated. Additional requirements, as specified in [11], are:

- Identity verification should not rely on the existence of a global PKI.
- Minimize the number of messages and bytes sent between the participating entities.
- Consider the computational capabilities of the MN's and CN's.
- Resist denial-of-service attacks.
- In any event, provide no weaker guarantees than IPv4.

The starting point for our investigation of authenticated binding updates is the realization that every authentication protocol relies on some form of previously established shared information. In general, an authentication protocol involves a *claimant* A and a *verifier* B. Presented with a purported identity of the claimant, the goal of the verifier is to verify the claimed identity. In the setting of Mobile IP, the IP address of an agent can be taken as its identity.

In order for claimant A to directly authenticate herself to verifier B, agent A must be able to perform some action, either individually or in collaboration with services available on the network, that can only be performed by A and that B is able to recognize, either individually or in collaboration with services available on the network. If A has a public-private key pair and the public verification key is known by B to be associated with A, then digital signatures allow B to verify a claim from A. Shared symmetric keys, shared secret hash keys, and other shared secret or unforgeable information can also be used for authentication. However, unless there is some kind of secret information, known to A and not known to a potential impersonator, and verifier B has prior knowledge that allows B to recognize some kind of operation involving the secret information held by A, there is no known way for B to reliably authenticate A.

Since any protocol that truly authenticates binding updates will depend on some form of chain of trust, our goal is to present the most reliable form of authentication possible for each of the most realistic or potentially achievable chains of trust. Further discussion on how our authentication protocol leverages various trust infrastructures appears in Section 3.7.

3.3. Previous Proposals

Two previous protocols for authenticating binding updates [13, 14] are susceptible to person-in-the-middle attacks. We review these protocols to show some of the forms of cryptography that have been considered and to illustrate their shortcomings.

Bradner, Mankin and Schiller [13] proposed a framework called *Purpose Built Keys*. Before initiating a connection with a correspondent node (CN), a mobile node (MN) generates a public-private key pair (called a purpose-built key pair) for use during the connection. The MN then sends a hash of the public key to CN. Subsequently, when MN sends a binding update, MN signs it with its private key and also sends the public key to CN. CN verifies that the public key hashes to the same value it had received before and, if so, uses the public key to verify the digital signature. An advantage of this framework is that it does not require any security infrastructure. However, purpose-built keys provide authentication if and only if the initial hash of the public key is received correctly by CN. This might not be the case. An attacker could intercept the hash and send the hash of a different key (which it owns) to CN. Subsequently, it can pretend to be MN without CN being any wiser. The authors of the draft were, of course, aware of this weakness.

Le and Faccin [14] propose two protocols for authenticating binding updates. The first assumes that both the MN and the CN share security associations with two AAA servers (e.g., RADIUS [25], DIAMETER [26]) and these two servers in turn share a security association. The protocol then uses this chain of trust to achieve authentication. Our protocol uses a similar architecture and can be easily adapted to work within a AAA infrastructure. Furthermore, it employs fewer messages than Le and Faccin's protocol (5 as opposed to 7). This improvement is obtained by combining encryption-based and signature-based authentication techniques and will be elaborated further in the next section. The second protocol proposed in [14] involves an unauthenticated Diffie-Hellman key exchange between MN and CN. The resulting key is subsequently used to authenticate binding updates. The authors recognize that this protocol is vulnerable to a man-in-the-middle (MITM) attack but state that "due to the properties of IP" such an attack will always be detected. They argue that since an attacker cannot remove any packets from the network, if a MITM attack is launched, both the MN and the CN will receive two Diffie-Hellman exponentials and will therefore be able to detect the attack. There are a couple of objections to this argument. Firstly, an attacker could remove packets from the network, e.g., if it gained control of the router being used by MN. Secondly, even if it were not able to remove packets from the network, it could delay them, e.g., by flooding MN and CN's network buffers with junk packets. During that interval, it could potentially do a lot of damage. Also, even if MN and CN detect the attack, the only thing they can do is drop the session. So, the attack still succeeds as a denial of service attack.

3.4. The Protocol

In this section, we present the authentication protocol based on our strongest assumptions. We assume that each mobile node (MN) has a pre-established bidirectional security association with its home agent (HA) using which they can authenticate each other. This is a reasonable assumption (and has been recognized as such in [11]) since typically the MN and its HA will belong to the same administrative domain. We also assume that HA's are capable of authenticating each other using public key cryptography.

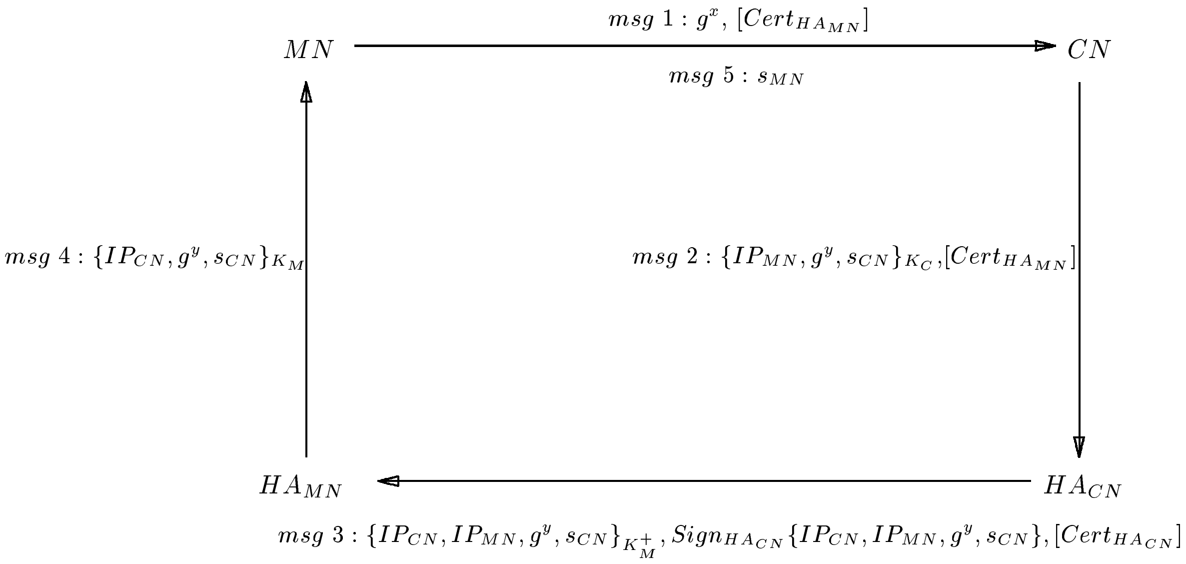

Figure 3.1. Initialization Phase

3.4.1 Notation

The following notation is used in describing the protocol.

MN	Mobile node
CN	Correspondent node
HA_{MN}	Home agent of MN
HA_{CN}	Home agent of CN
BU	Binding update
K_M	Shared secret between MN and HA_{MN}
K_C	Shared secret between CN and HA_{CN}
K_M^+	Public encryption key of HA_{MN}
$Cert_i$	Certificate of entity i
$Sign_i\{\ldots\}$	Signed by entity i
$\{\ldots\}_K$	Encrypted with key K
$[x]$	x is an optional parameter
IP_i	IP (home) address of i
K_D	Shared Diffie-Hellman secret key
$h(k, M)$	Keyed hash of message M with key k
s_{MN}	$\{IP_{MN}, IP_{CN}, g^x, g^y\}_{K_D}$
s_{CN}	$\{IP_{CN}, IP_{MN}, g^y, g^x\}_{K_D}$

3.4.2 Protocol Description

Our protocol consists of two phases: (a) an *initialization phase* in which MN and CN set up an authentication key; (b) an *update phase* in which MN sends an authenticated binding update to CN using the key obtained from phase (a).

Initialization Phase

The initialization phase is an authenticated Diffie-Hellman key exchange protocol [27] resulting in a shared secret between an MN and a CN. The protocol is shown in Figure 3.1. We first take a closer look at the protocol and discuss the purpose served by the different message components. Then, we describe some of the features of this protocol.

Message 1 is straightforward. Note that it assumes that MN already knows a group and generator that is acceptable to CN.

Upon receiving message 1, CN generates g^y, computes $K_D = g^{xy}$ and s_{CN} and sends a message to HA_{CN} encrypted with their shared key, K_C. Note that the use of encryption-based authentication ensures that no attacker can observe g^y. s_{CN} consists of the session parameters encrypted with the session key, K_D. It is forwarded by the two home agents and proves to MN that CN has computed the same secret key. Thus, the protocol provides *direct authentication*, a desirable property as described in [29]. If direct authentication is not required, then s_{CN} can be replaced by g^x in messages 2, 3 and 4 resulting in a lighter-weight protocol.

In Message 3, HA_{CN} forwards to HA_{MN} the session parameters that it recovers by decrypting message 2. It is necessary to include $CN's$ IP address since the same MN might be executing this protocol parallely with multiple correspondent nodes. Encrypting the session parameters with public key of HA_{MN} ensures that no other entity can recover g^y. The signature proves to HA_{MN} that the message originated from HA_{CN}. Note that the signature scheme should be such that no information regarding plaintext data can be deduced from the signature itself on that data (e.g., when the signature operation involves preliminary one-way hashing). This is critical because, in general, data may be recovered from a signature on it (e.g., RSA without hashing). An alternative approach would be to include the signed block inside the public key encryption. A disadvantage of this method is that here the data to be public-key encrypted is larger. This might require adjustment of the block-size of the public encryption scheme or the use of techniques like cipher-block-chaining (see Section 12.5.2 of [28] for further discussion on the relative advantages of the two methods).

HA_{MN} forwards the session parameters to MN in message 4, encrypted with K_M, their shared key. Encryption serves the same purpose as before: it hides g^y from all other entities. We note here that the structure of the plaintext encrypted with K_M in this message is quite similar to that encrypted with K_C in message 2. While this does not lead to any attack under the assumption of black box cryptography, it might be useful to break the symmetry by reordering two message components, e.g., g^y and s_{CN} in message 4. It has been recognized that symmetries in a protocol should be used with caution, due to both the possibility of reflection attacks, and attacks in which responses from one party can be reused within a protocol (see [29]).

In message 5, MN encrypts the session parameters with the newly computed Diffie-Hellman secret. This serves two purposes. Firstly, it proves to CN that the message actually came from MN (since the only entities which know g^y are CN, HA_{CN}, HA_{MN}, and MN and we assume that these entities are honest). Secondly, it provides proof that both MN and CN have computed the same secret, i.e., it provides direct authentication.

In this protocol, CN authenticates herself to MN through the chain of trust $CN \rightarrow HA_{CN} \rightarrow HA_{MN}$ $\rightarrow MN$. MN could also use the same trust chain in the other direction to authenticate herself to CN. This would lead to a 7-message protocol similar to the one presented in [14]. In order to reduce the number of messages to 5, we use a different technique. $CN's$ Diffie-Hellman exponential is never sent in the clear; it is sent encrypted along the trust chain and the very fact that MN is able to recover it and compute the same Diffie-Hellman secret as CN serves to verify her identity. We thus combine the two standard techniques of signature-based and encryption-based authentication to realise a protocol with fewer messages. The difference in the number of messages exchanged in the two protocols becomes more appreciable as the trust chain becomes longer. If there are n participants, our protocol requires $n+1$ messages while the previous approach requires $2n-1$. It also appears that we can do no better. Specifically, any Diffie-Hellman based authenticated key exchange protocol P between entities A and B that uses a chain of trust of length n and provides direct authentication requires at least $n+1$ messages.

The use of Diffie-Hellman key exchange ensures that the protocol provides *perfect forward secrecy*

(PFS), i.e., the disclosure of long-term secrets like private signing/decryption keys does not compromise the secrecy of exchanged keys from earlier runs. However, if perfect forward secrecy is not a requirement, an alternative would be to replace the exchange of DH exponentials by an exchange of nonces (n_{MN} and n_{CN}). n_{MN} could be sent in the clear in message 1, while n_{CN} is sent encrypted in message 2, 3, 4. The shared secret could be derived from a hash of these nonces, e.g., HMAC(n_{MN}, n_{CN}) (see [30]). Note that this variation of the base protocol does not have PFS since the compromise of long term secrets like the shared key between CN and HA_{CN} exposes all past n_{CN} values and hence all past session keys. The advantage is that it is computationally very lightweight: mobile nodes have to perform relatively inexpensive operations - generating a random nonce and computing a hash instead of exponentiation.

Typically, in order to prevent replay attacks, key exchange protocols require each participant to use some fresh information in every run of the protocol. The Diffie-Hellman exponentials serve this purpose in our protocol. We would also like to note here that an implicit assumption in our protocol is that it is possible to verify whether a home agent actually owns a mobile node which it claims as its own (e.g., by matching the network prefix of the home address of the mobile node with that from the home agent's certificate). Otherwise, the protocol is open to attack. Any adversary which possesses a certificate could intercept message 1 and then send message 3 without HA_{MN} being any wiser.

Update Phase

Once MN and CN have set up a shared secret, K_D, MN can easily send an authenticated binding update (BU) by executing the following 1-message protocol.

$$BU, h(K_D, BU)$$

Here, $h(\ldots)$ is a keyed cryptographic hash function (e.g., HMAC [30]). This protocol provides sender authentication since MN is the only entity other than CN which possesses the key K_D. Data integrity is also provided since the hash is also a message authentication code (MAC). Replay attacks can be prevented by using the *sequence number* field in the binding update option (see Section 5.1 of [12]). The sequence number field holds an 8-bit number. Each binding update sent by a mobile node must use a sequence number which is greater than the sequence number of the previous binding update sent to the same destination address. Every time the sequence number space is exhausted, the shared key should be refreshed. Key refresh could, of course, be done by re-executing the initialization phase protocol.

Alternatively, MN and CN could set up a new key by executing an authenticated Diffie-Hellman key exchange protocol in which they can use the old key to authenticate messages. This approach would require the home agents to be involved in only the initialization phase (once per (MN, CN) pair) and the system would scale up better.

3.5. Analysis of the Protocol

We used Murφ, a finite-state analysis tool, to carry out a formal analysis of the initialization phase of our protocol. In this section, we describe some of the challenges we faced in modelling this protocol.

3.5.1 Modelling the Initialization Phase

The Murφ model of the protocol consists of three types of finite state machines corresponding to mobile nodes, home agents and intruders. The number of mobile nodes, home agents and intruders are scalable and defined by constants. Each mobile node can participate in a number of parallel sessions. This number can also be configured by changing the value of a constant.

The state of a mobile node consists of her IP address, the IP address of her home agent, the secret she shares with her home agent and the individual states of all the sessions that she is involved in. We associate the *state-id* of a session with the next message that the node is expecting in that session. We denote by S_i the state in which a node is expecting the i^{th} message of the protocol. For example, after initiating a session (sending message 1), a node sets the state-id of that session to S_4 since the next message that it expects is the 4^{th} message of the protocol. Initially, the state-ids of all sessions are set to S_1. In this state, a mobile node can spontaneously initiate a session. Other than the state-id, the state of a session also includes the values of all the session parameters that the mobile node has seen uptil that point. For example, if the state-id of a session that a mobile node is taking part in is S_4, then the state would also contain the node's Diffie-Hellman private key and the address of the peer node with whom she is executing the session. Upon completing a session, the state-id for that session is set to S_{DONE}. The transition rules for a mobile node capture the exact sequence of actions that she would carry out in an actual run of the protocol. For example, when a mobile node receives the 4^{th} message of the protocol, she processes it iff the corresponding state-id is S_4. She then verifies that the encryption key specified in the message is the same as the key she shares with her home agent. This corresponds to decrypting the message. Then she computes the Diffie-Hellman secret using the Diffie-Hellman exponential in the message and her own previously recorded Diffie-Hellman private key. She finally verifies that the computed secret matches the one in the message before changing the state-id to S_{DONE}.

The finite state machine of a home agent is much simpler. Since a home agent only forwards messages, his state does not change during the execution of the protocol. The state of a home agent consists of his certificate, the verification key of the trusted third party (Certification Authority of the PKI) and the shared keys with the mobile nodes in his domain. The transition rules define the sequence of actions that a home agent executes upon receiving the 2^{nd} or 3^{rd} message of the protocol.

As mentioned before, the intruder's transition rules enable it to intercept messages, overhear all messages and remember parts of all overheard messages and generate new messages using any combination of initial knowledge and parts of overheard messages. Thus, at any given point in the protocol, there are a large number of possible transitions for the intruder. This results in a very large number of reachable states for the protocol. The following techniques proved useful in reducing the number of states to be explored:

- The intruder always intercepts all messages sent by the honest participants.
- The intruder does not send messages to honest participants in states where at least one of the honest participants is able to send a message.
- The intruder only generates messages that are expected by the legitimate parties and that can be meaningfully interpreted by them in their current state, e.g., the intruder sends the 4^{th} message to a mobile node only if the mobile node is in state S_4.

The first two techniques have been proved to be sound (see [33]), i.e., each protocol error that would have been discovered in the original state graph will still be discovered in the reduced state graph. The soundness of the third technique is quite obvious.

We modelled the following correctness conditions in our Murφ code:

- If two mobile nodes have completed a session with each other, then the shared Diffie-Hellman secret computed by both must be identical.

- The secret shared between two mobile nodes is not in the intruder's database of known message components.
- The mobile nodes agree on each other's identity and protocol completion status, i.e., if A has completed a session with B, then B should also have completed the same session with A or B should be waiting for the 5^{th} message of the protocol.

Murφ did not discover any violations of the above mentioned correctness conditions for the configurations on which we ran the verifier. Running under Linux on a 300 MHz dual-processor Pentium II with 512MB of RAM, the verifier required approximately 30 seconds to check for the case with 2 mobile nodes, 2 home agents and no more than 2 simultaneous sessions per mobile node. About 4200 states were explored. The largest instance of our model that we verified included 3 mobile nodes, 3 home agents and no more than 2 simultaneous sessions per mobile node. Checking took about 20 minutes, with 125,941 states explored.

We note here that this is the first Diffie-Hellman key exchange based protocol that has been modelled using Murφ. Modelling the DH-protocol within the finite state analysis framework required some thought. The "obvious" approach would be to do what is actually done in practice: use integers to explicitly model the various parameters (g, p, x, y) and then derive the secret by exponentiating modulo p. However, we observe that explicit exponentiation is unnecessary. The two main properties that the model should capture are: (a) the computational hardness of the Diffie-Hellman problem, i.e., given $g^x \ mod \ p$ and $g^y \ mod \ p$, it should not be possible to compute $g^{xy} \ mod \ p$; (b) the commutativity of exponentiation so that $(g^x \ mod \ p)^y \ mod \ p = (g^y \ mod \ p)^x \ mod \ p$. (From Fermat's Little Theorem, we also know that both these values are equal to $g^{xy mod (p-1)} \ mod \ p$. In order to capture these two properties we used a different technique. In the Murφ code, a *key* is modelled as a record with two fields - a *type* and a *random value*. We defined three additional types of keys: *dh-private*, *dh-public*, and *dh-secret*. x is of type dh-private and g^x is of type dh-public. For both keys, the value is x. The DH secret is computed using a function which takes in a key of type dh-private (say with value x) and another of type dh-public (say with value y) and returns a key with type dh-secret and value $xy \ mod \ (p-1)$. Note that interchanging the values of x and y yields the same secret because of the commutativity of integer multiplication. This ensures that both participants compute the same Diffie-Hellman secret. Also, since this function is the only way of computing a DH secret, property (a) above is also satisfied.

3.6. Preventing Denial-of-service Attacks

The basic 5-message protocol is susceptible to denial of service attacks. By sending a random number to a CN, an attacker can force a CN to perform a Diffie-Hellman exponentiation and an encryption operation. The CN will also create state at this point. By continuously initiating sessions with a CN, an attacker can exhaust its computation and memory resources. One way of preventing this attack would be to use "cookies", a technique originally proposed by Karn and Simpson in [34]. Upon receiving the first message, CN replies with a cookie (which could be a keyed hash of the received exponential concatenated with a timestamp and the IP address of the sender as used in the IKE-SIGMA protocol [35]). The sender then sends the cookie back to CN proving that it is capable of receiving messages at the IP address it is claiming as its own. Thus, two additional messages are exchanged between MN and CN after the first message of the original protocol.

A useful property of our protocol is that since home agents do not create state, memory denial of service attacks are not possible on the home agents. Preventing computation denial of service attacks on the home agents reduces to the problem of detecting, without performing expensive computations, whether a message has been replayed. Replay attacks on HA_{CN} can be prevented by including a sequence number or timestamp inside the encryption in message 2 of the protocol. HA_{CN} will accept a message from CN only if the sequence number is greater than the last sequence number received from the same CN. Preventing denial of service attacks on HA_{MN} without adding extra messages, appears to be more difficult. Adding a sequence number to message 3 and including it in the signature alleviates the problem somewhat. A replayed message will be detected after performing one public key operation instead of two. Adding two extra messages for exchanging cookies, of course, solves the problem.

3.7. Trust Infrastructure and Authentication

The protocol in Section 3.4 relies on the existence of a PKI. However, it can be easily adapted to work in any infrastructure which provides a chain of trust between mobile and correspondent nodes. In particular, if a AAA infrastructure [25, 26] is deployed in which MN and CN can authenticate themselves to AAA_{MN} and AAA_{CN} respectively, and the two servers share a secret K_A, the only change required is to let the AAA servers take on the role of the home agents and replace message 3 of the original protocol by the following message:

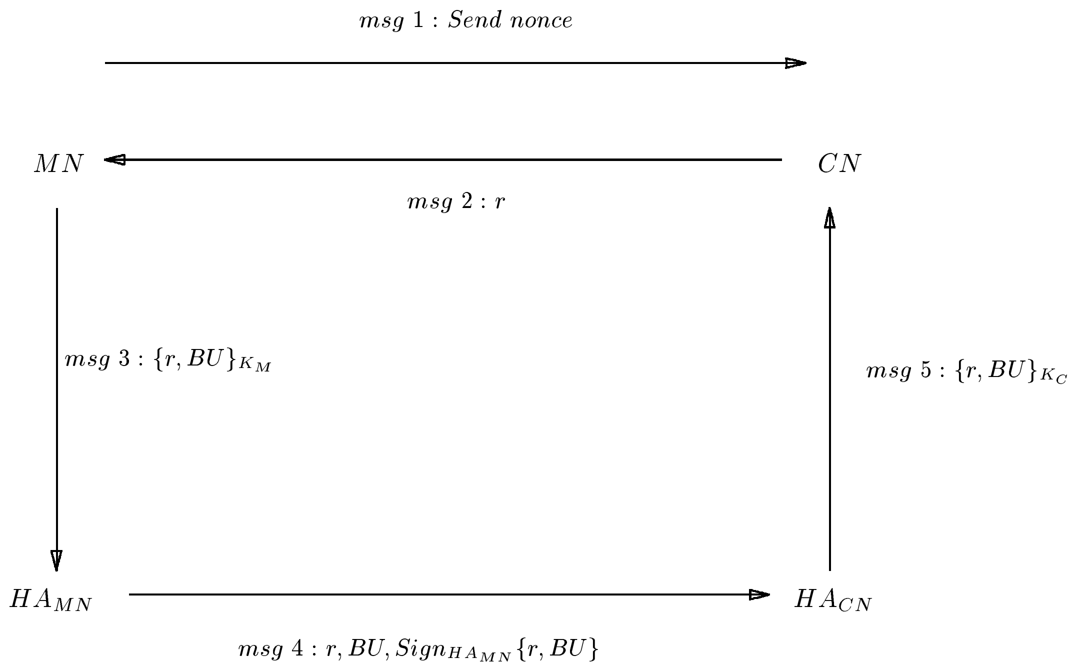

Figure 3.2. Protocol for Unidirectional Trust Chain

$$AAA_{CN} \rightarrow AAA_{MN}: \quad \{IP_{CN}, IP_{MN}, g^y, s_{CN}\}_{K_A}$$

In the remainder of this section, we consider four alternative trust infrastructures and describe how authentication can be achieved in each one. In what follows, we assume throughout that MN and CN can authenticate themselves to HA_{MN} and HA_{CN} respectively. We distinguish between the following four trust infrastructures:

1. *MN and CN can directly authenticate each other.*

   A special case is when MN and CN share a secret key. It is then possible for MN to send an authenticated binding update to CN by executing the update phase protocol of Section 3.4.2. This protocol does not require the involvement of any third party and is also computationally inexpensive. Another case would be when both MN and CN possess signing-verification key pairs. However, this option is not too attractive since public key cryptography is computationally intensive and may be beyond the capabilities of mobile nodes.

2. *There exists a bidirectional chain of trust between MN and CN.*

   An example is when HA_{MN} and HA_{CN} can authenticate each other using public key cryptography. The bidirectional chain of trust can be used to set up a shared secret between MN and CN by executing the initialization phase protocol of Section 3.4.2. This protocol is more expensive than the previous one: it involves home agents and requires them to perform public key operations. However, the initialization phase needs to be executed only once per $MN - CN$ pair. Once executed MN and CN end up with a shared secret and can directly authenticate each other.

3. *There exists a unidirectional chain of trust from MN to CN.*

   Consider, for example, the case when HA_{MN} can authenticate herself to HA_{CN} but not vice versa. MN can then use the trust chain $MN \rightarrow HA_{MN} \rightarrow HA_{CN} \rightarrow CN$ to send an authenticated binding update as shown in the protocol in Figure 3.2. This protocol provides sender authentication and data integrity. In message 1, MN requests CN to send a nonce. The nonce, r, sent in message 2 is used to prevent replay attacks. We note, however, that this protocol places higher computation demands on the home agents. The home agents have to perform public key operations everytime a binding update is sent. It is not possible to set up a shared secret between MN and CN since CN has no way of authenticating herself to MN.

4. *There is no chain of trust from MN to CN.*

   In the absence of a chain of trust from MN to CN, it does not appear that strong authentication is possible. However, we can design a protocol with a narrower window of vulnerability than one which provides no authentication at all. One way to achieve this would be for two home agents to set up a shared secret the first time they need to communicate with each other. This could be done, e.g., by an unauthenticated Diffie-Hellman key exchange. Subsequently, they can authenticate each other using this key. Thus, assuming that this operation is performed securely, all future transactions between the nodes of these home agents can be secured. A similar security property could be obtained by exchanging public keys without having them certified by a Certification Authority.

We observe that there is a clear connection between infrastructural support and the computational requirements of the corresponding authentication protocol: as we relax our assumptions about the infrastructure, protocol participants have to do a greater amount of computation to complete the authentication process. The amount of computation gradually increases from case 1 to 3 in the list above. We also note that if a home agent possesses a public signing key, then all her mobile nodes can send authenticated binding updates even if the corresponding node's home agent does not possess a public key. This gives some incentive for a home agent to acquire a public key certificate. However, if the corresponding node's home agent does not possess a public key, the mobile node's home agent has to perform public key operations everytime a binding update is sent. This raises the issue of scalability. If a home agent is serving a number of mobile nodes, its computational resources might be exhausted. The trust infrastructure presented in case 2 therefore seems to be best for this application.

3.8. Conclusions

In response to the requirement that all location information about a mobile node in IPv6 should be authenticated [12, 3.1], we have proposed a protocol for authenticating binding updates. The other requirements are taken into account as well: the computational load on the nodes and on the routers is minimized, by eliminating expensive encryption operations and keeping the number of messages at minimum. The appropriate extensions preventing denial-of-service attacks are also suggested.

We have formally verified the correctness of the protocol using the finite-state analysis tool Murφ. The fact that the Murφ analysis did not uncover any bugs gives us increased confidence in the correctness of the protocol. In previous work, Murφ has been used to analyze the security properties of several protocols. However, this is the first Diffie-Hellman based key exchange protocol that has been analyzed with Murφ. We used a new modelling technique to capture the two most important properties of the Diffie-Hellman exchange: the computational hardness property which ensures that an intruder is not able to compute the DH secret from the public exponentials; and the commutativity property which guarantees that both participants compute the same shared secret.

Finally, we have addressed the most difficult adoption issue for authenticated Mobile IPv6: reliance on authentication infrastructure. Besides the base protocol which assumes the existence of a bidirectional trust chain between mobile and correspondent nodes, we considered three other trust infrastructures and presented variations of the initialization phase that are appropriate to each scenario. The update phase of our protocol remains the same in each case since the basis for authenticated update is the shared secret established in the initialization phase. We believe that our protocol addresses the main issues in Mobile IPv6 authentication and makes best use of whatever infrastructure is available. In particular, the SSH-like scheme, using unauthenticated key exchange between home agents the first time these home agents participate in Mobile IP, might be the most practical one in the near term.

Chapter 4

Analysis of the JFK Protocol

4.1. Introduction

JFK [36] is a protocol recently proposed to replace IKE [37] 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 chapter, 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 [27] and a simplified version of the standard challenge-response authentication protocol [28] 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.

In addition, we have also formally verified the correctness of the JFK protocol using the finite state analysis tool Murφ [17]. While formal methods have been successfully used to analyze key exchange and authentication protocols [18, 19, 20, 21, 3, 6], the analysis of JFK required several new concepts and modelling techniques. In particular, modelling the Diffie-Hellman key exchange required some thought. This is the first key exchange protocol that allows reuse of Diffie-Hellman exponentials across multiple sessions and this feature also had to be explicitly modelled. While the key secrecy and mutual authentication properties have been verified, we are currently in the process of verifying the other stated design goals of the protocol, particularly, its resistance to DoS attacks.

The remainder of this chapter is structured as follows. Section 4.2 describes the the design goals of the JFK protocol. Section 4.3 discusses the protocol itself and the role served by the different message components. In Section 4.4, we present a systematic reconstruction of the core JFK protocol using compositions, refinements and transformations. Section 4.5 presents a protocol transformation technique which adds DoS protection to the base protocol. In Section 4.6, we present our modelling assumptions and analysis results. Concluding remarks and directions for future work appear in Section 4.7.

4.2. 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 [34] 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.

4.3. The JFK Protocol

4.3.1 Notation

The following notation is used in describing the protocol.

$A \to B$	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

4.3.2 Protocol Description

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

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

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

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

Figure 4.1. JFK Protocol

The JFK protocol is shown in Figure 4.1. 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 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).

4.4. 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 [27] 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 [28]).

   $I \rightarrow R : m$

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

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

2. Transformation:

   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.

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. 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 [38]), 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 [2].

   (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 [28] (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 refinement 3(c) to the protocol obtained from Step 4.

  $I \rightarrow R : g^i$

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

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

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

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

- Step 6: Apply refinement 3(d) to the protocol obtained from Step 5.

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

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

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

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

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

- Step 7: Since I needs to know ID_r before she signs it in message 3, ID_r is moved from message 4 in Step 6 to message 2 here.

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

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

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

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

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

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.

4.5. 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.

4.5.1 Protocol Description Notation

We use an extension of the strand space notation [39] 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:

$I \rightarrow R : g^i$

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

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

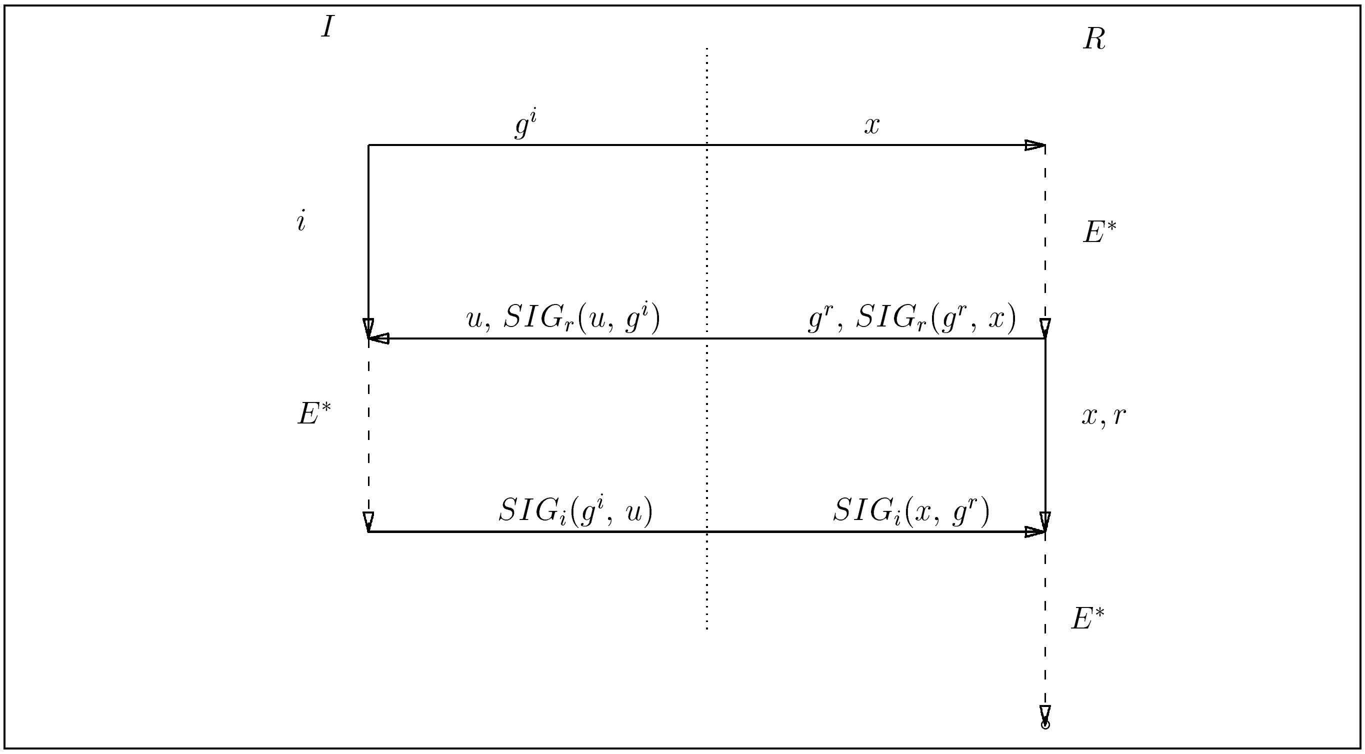

Figure 4.2. Example

The representation of this protocol in the extended strand space notation is shown in Figure 4.2. 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.

4.5.2 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 4.3. 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 4.4. 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).

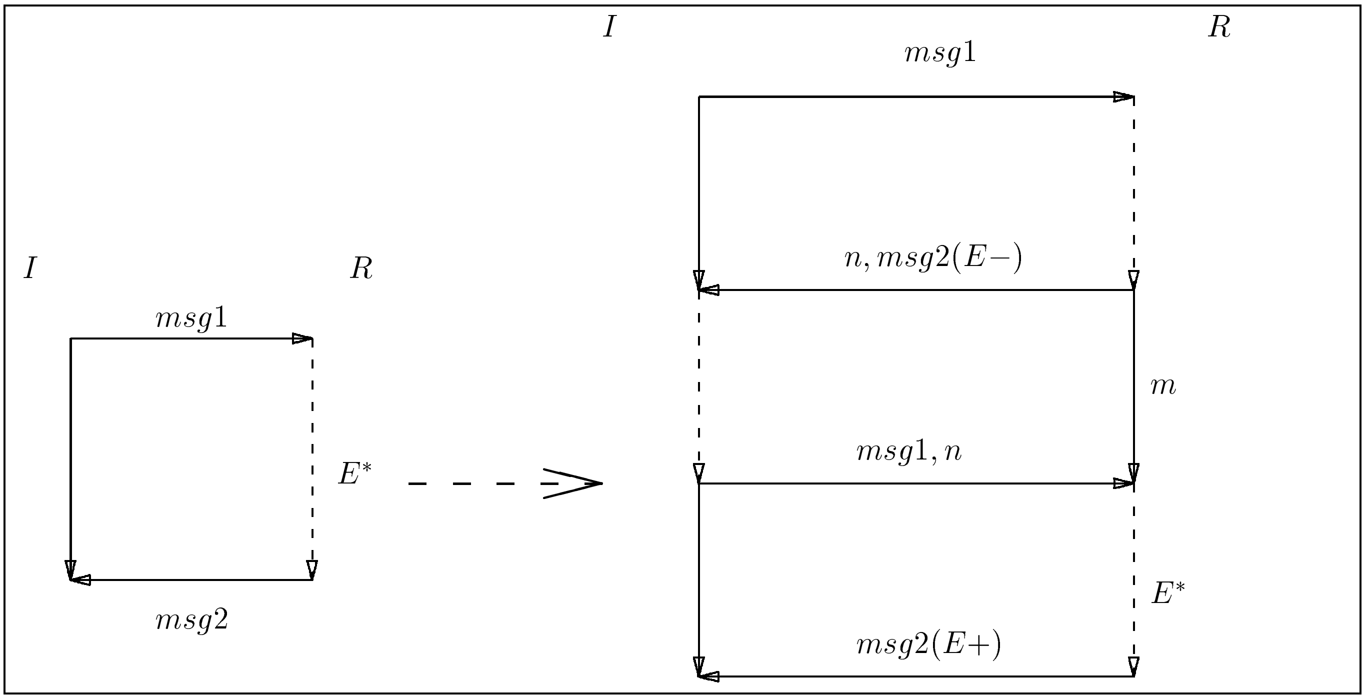

Figure 4.3. CDoS Protection Transformation

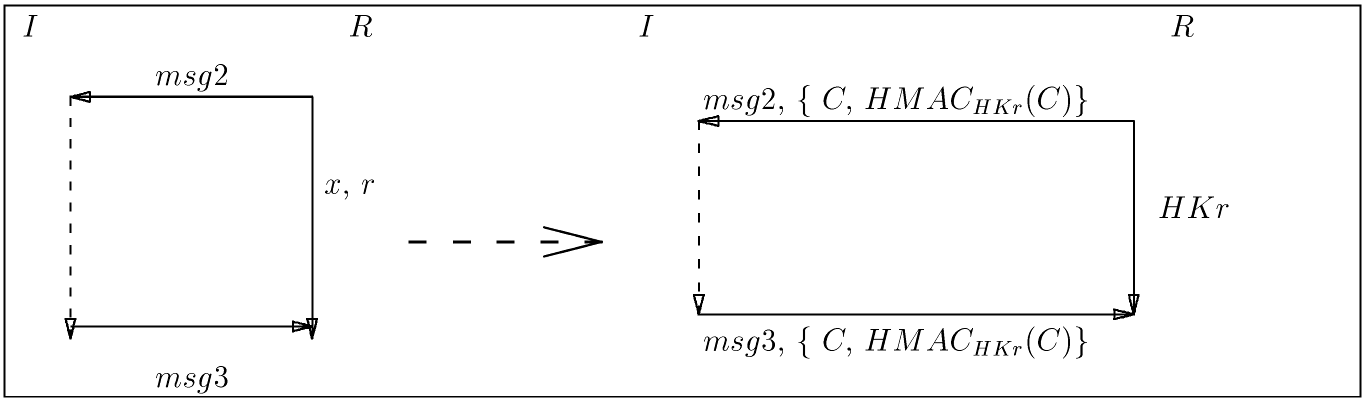

Figure 4.4. MDoS Protection Transformation

3. CDoS and MDoS protection:

   We compose the transformations of Figure 4.3 and Figure 4.4 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 4.5. 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 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.) We can apply this rule to the protocol of Figure 4.5 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 4.6. It is identical to the one obtained from Step 2 in the previous section.

4.6. Murφ Analysis

We used Murφ, a finite-state analysis tool, to carry out a formal analysis of JFK. In this section, we describe some of the challenges we faced in modelling this protocol.

4.6.1 Modelling JFK

The Murφ model of the protocol consists of two types of finite state machines corresponding to protocol participants and intruders. The number of protocol participants and intruders are scalable and defined by constants. Each participant can engage in a number of parallel sessions. This number can also be configured by changing the value of a constant.

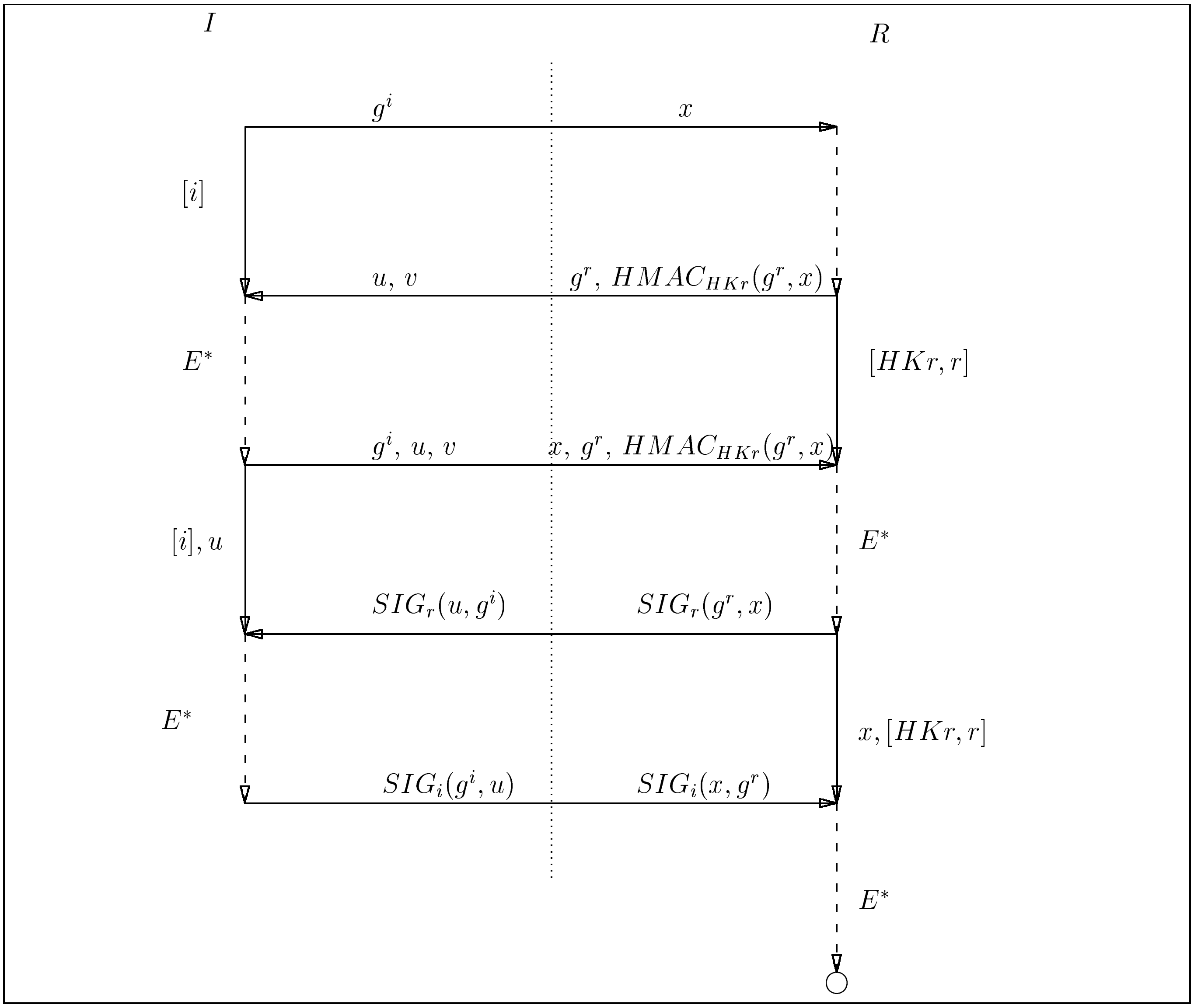

Figure 4.5. DoS Protection Transformation

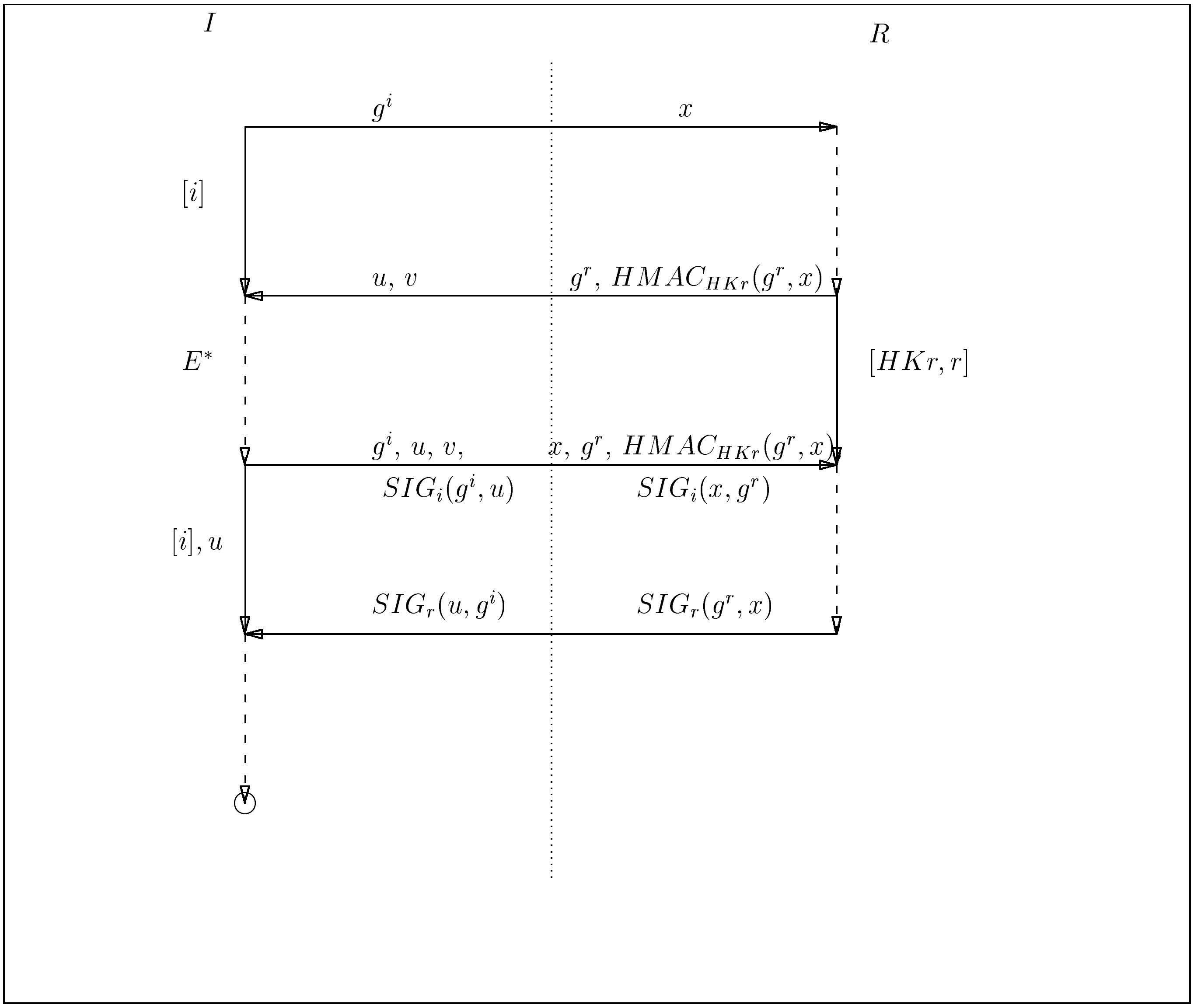

Figure 4.6. Optimized Transformed Protocol

We assume that each process can be an Initiator for one session and a Responder for another. Each process maintains state for each of the sessions that it is involved in. We associate the *state-id* of a session with the next message that the process is expecting in that session. We denote by S_i the state in which a process is expecting the i^{th} message of the protocol.Note, however, that since the Responder does not create state after responding to message 1, it is in state S_1 and not S_3 when it receives message 3. Other than the state-id, the state of a session also includes the values of all the session parameters that the process has seen uptil that point. The transition rules for a process capture the exact sequence of actions that she would carry out in an actual run of the protocol.

As mentioned before, the intruder's transition rules enable it to intercept messages, overhear all messages and remember parts of all overheard messages and generate new messages using any combination of initial knowledge and parts of overheard messages. Thus, at any given point in the protocol, there are a large number of possible transitions for the intruder. This results in a very large number of reachable states for the protocol. The following techniques proved useful in reducing the number of states to be explored:

- The intruder always intercepts all messages sent by the honest participants.
- The intruder does not send messages to honest participants in states where at least one of the honest participants is able to send a message.
- The intruder only generates messages that are expected by the legitimate parties and that can be meaningfully interpreted by them in their current state, e.g., the intruder sends the 4^{th} message to a process only if the process is in state S_4.

The first two techniques have been proved to be sound (see [33]), i.e., each protocol error that would have been discovered in the original state graph will still be discovered in the reduced state graph. The soundness of the third technique is quite obvious.

We modelled the following correctness conditions in our Murφ code:

- If two processes have completed a session with each other, then the shared Diffie-Hellman secret computed by both must be identical.
- The secret shared between two processes is not in the intruder's database of known message components.

- The processes agree on each other's identity and protocol completion status, i.e., if A has completed a session with B, then B should also have completed the same session with A or B should be waiting for the 4^{th} message of the protocol.
- The same key is not used in more than one session by a process. This condition ensures that if an Intruder gets to know one session key, it cannot force the Responder to set up another session with the key.

Murφ did not discover any violations of the above mentioned correctness conditions for the configurations on which we ran the verifier. Running under Linux on a 300 MHz dual-processor Pentium II with 512MB of RAM, the verifier required approximately 20 minutes to check for the case with 2 protocol participants, and no more than 3 simultaneous sessions per participant. About $125,701$ states were explored.

We faced several challenges in carrying out the Murφ analysis of the protocol. Firstly, modelling the DH-protocol within the finite state analysis framework required some thought. The "obvious" approach would be to do what is actually done in practice: use integers to explicitly model the various parameters (g, p, x, y) and then derive the secret by exponentiating modulo p. However, we observe that explicit exponentiation is unnecessary. The two main properties that the model should capture are: (a) the computational hardness of the Diffie-Hellman problem, i.e., given $g^x \ mod\ p$ and $g^y \ mod\ p$, it should not be possible to compute $g^{xy} \ mod\ p$; (b) the commutativity of exponentiation so that $(g^x \ mod\ p)^y \ mod\ p$ $= (g^y \ mod\ p)^x \ mod\ p$. (From Fermat's Little Theorem, we also know that both these values are equal to $g^{xymod(p-1)} \ mod\ p$. In order to capture these two properties we used a different technique. In the Murφ code, a *key* is modelled as a record with two fields - a *type* and a *random value*. We defined three additional types of keys: *dh-private*, *dh-public*, and *dh-secret*. x is of type dh-private and g^x is of type dh-public. For both keys, the value is x. The DH secret is computed using a function which takes in a key of type dh-private (say with value x) and another of type dh-public (say with value y) and returns a key with type dh-secret and value $xy \ mod\ (p-1)$. Note that interchanging the values of x and y yields the same secret because of the commutativity of integer multiplication. This ensures that both participants compute the same Diffie-Hellman secret. Also, since this function is the only way of computing a DH secret, property (a) above is also satisfied.

JFK allows participants to use the same Diffie Hellman key in multiple sessions within a time interval. In order to model this, we maintained a *mod n* counter at every process. Every time a process completes a session, it increments the counter $mod\, n$. When the counter value returns to zero, a new Diffie Hellman key is generated. Also, the key HKr is refreshed.

The Responder keeps a copy of recently received message 3's and their corresponding message 4's. Receiving a duplicate (or replayed) message 3 causes the Responder to simply retransmit the corresponding message 4 without creating new state. This is necessary to guarantee that two different sessions do not generate the same key. In our model, in order to reduce the size of the state space, we do not create an explicit cache for message $< 3, 4 >$ pairs. The key observation is that when a Responder receives a duplicate message 3, it can reconstruct the corresponding message 4 using the contents of message 3 and the history of the original session. We use a boolean variable 'inCache' to keep track of whether the message $< 3, 4 >$ pair is currently in cache. In accordance with the specification, the cache is reset whenever the Diffie Hellman key and HKr is refreshed.

4.7. 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 [27] and a simplified version of the standard challenge-response authentication protocol [28], 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.

In addition, we have also formally verified the key secrecy and authentication properties of the JFK protocol using the finite state analysis tool Murφ [17]. Although in previous work, Murφ has been used to analyze the security properties of several protocols, the analysis of JFK required several new concepts and modelling techniques. In particular, modelling the Diffie-Hellman key exchange required some thought.

We used a new technique to capture the two most important properties of the Diffie-Hellman exchange: the computational hardness property which ensures that an intruder is not able to compute the DH secret from the public exponentials; and the commutativity property which guarantees that both participants compute the same shared secret. Also, this is the first key exchange protocol that allows reuse of Diffie-Hellman exponentials across multiple sessions and this feature also had to be explicitly modelled. While the key secrecy and mutual authentication properties have been verified, future work would include verifying the other stated design goals of the protocol, particularly, its resistance to DoS attacks. At this point, our analysis seems to indicate that JFK is founded on sound protocol design principles.

Bibliography

[1] D. Dolev, A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198-208, 1983.

[2] 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.

[3] 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.

[4] Z. Dang, R. A. Kemmerer . Using the ASTRAL Model Checker for Cryptographic Protocol Analysis. *In Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[5] B. Dutertre, S. Schneider . Using a PVS Embedding of CSP to verify Authentication Protocols. *In TPHOLS'97*, 1997.

[6] L. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *In Journal of Computer Security*, 6:85-128, 1998.

[7] A. Huima. Efficient Infinite-state Analysis of Security Protocols. *In Proc. Workshop on Formal Methods and Security Protocols*, 1999.

[8] W. Marrero. A Model-checker for Authentication Protocols. *In Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[9] D. Song. Athena: An Automatic Checker for Security Protocol Analysis. *In Proc. IEEE Computer Security Foundations Workshop*, 1999.

[10] A. Durante, R. Focardi, R. Gorrieri. CVS: A Compiler for the Analysis of Cryptographic Protocols. *In Proc. IEEE Computer Security Foundations Workshop*, 1999.

[11] A. Mankin, B. Patil, D. Harkins, E. Nordmark, P. Nikander, P. Roberts, T. Narten. Threat Models introduced by Mobile IPv6 and Requirements for Security in Mobile IPv6. *Internet Draft*, October 2001.

[12] D. Johnson, C. Perkins. Mobility Support in IPv6. *Internet Draft*, July 2001.

[13] S. Bradner, A. Mankin, J.I. Schiller. A Framework for Purpose Built Keys (PBK). *Internet Draft*, February 2001.

[14] F. Le, S.M. Faccin. Dynamic Diffie Hellman based Key Distribution for Mobile IPv6. *Internet Draft*, April 2001.

[15] J. M. Galvin. Public Key Distribution with Secure DNS. *In Proc. 6th USENIX Unix Security Symposium*, 1996.

[16] G. Ateniese, S. Mangard. A New Approach to DNS Security. *In Proc. 8th ACM Conference on Computer and Communications Security*, 2001.

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

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

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

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

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

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

[23] 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.

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

[25] C. Rigney, A. Rubens, W. Simpson, S. Willens. Remote Authentication Dial In User Service (RADIUS). *RFC 2865*, June 2000.

[26] P. R. Calhoun, H. Akhtar, J. Arrko, E. Guttman, A.C. Rubens, G. Zorn. Diameter Base Protocol. *Internet Draft*, November 2001.

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

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

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

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

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

[32] 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.

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

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

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

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

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

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

[39] 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.