N00014-01-C-0454

Submitted to NDSS

Authenticated Address Notification for Mobile Communication

Anupam Datta John C. Mitchell Frederic Muller
Dept. Computer Science, Stanford University
{danupam,jcm,fmuller}@cs.stanford.edu

Dusko Pavlovic
Kestrel Institute, Palo Alto, CA
dusko@kestrel.edu

Abstract

We present an improved protocol for authenticating Mobile IPv6 connections, addressing requirements established in the relevant Internet Draft [3]. 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 takes advantage of available authentication infrastructure to set up a shared secret between a mobile and correspondent node. Each execution of the update phase uses the shared secret established in the previous phase to maintain security of the mobile connection. We have formally verified the correctness of the protocol using the finite-state analysis tool Murφ, which has been used previously to analyze hardware designs and security properties of several protocols.

1. Introduction

In Mobile IPv6 [1], 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. The need for authenticating address notification messages has been recognized [1, 3]. While providing authentication, any proposed solution must address two additional concerns: (a) computational tasks should be appropriately distributed so that smaller mobile nodes do not need to perform tasks which require expensive computations; (b) the available authentication infrastructure may be limited in the sense that some nodes may possess public key certificates while others may not be part of a global public key infrastructure. Previous proposals either fall short of meeting the authentication requirements [1, 4, 5] or use IPSec thereby requiring mobile nodes to possess public key certificates from a global PKI and perform expensive computation [2].

In this paper, we present a protocol for authenticating address notification messages (or *binding updates* in Mobile IPv6 terminology), taking into consideration the computational and infrastructure requirements. Throughout, we use the Mobile IPv6 scenario as a concrete example. We emphasize, however, that the proposed protocol would be applicable in other situations where mobile devices communicate with each other. In particular, cellular phone networks could provide another useful application scenario.

The protocol has two phases: (a) a once-per-connection *initialization phase* in which a mobile node and a correspondent node use any available chain of trust through authentication servers in their respective domains to confirm a shared secret; (b) an *update phase* that is executed every time an authenticated address notification message needs to be sent; the shared secret established in the previous phase is used as the basis of authentication. 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 correspondent node, and a reply through the authentication servers only once in the initialization phase, the protocol minimizes the number of messages exchanged between participating entities. While we assume that mobile and correspondent nodes share a secret key with their respective authentication servers, the protocol does not require the servers to maintain state during the execution of the protocol, avoiding memory denial-of-service attacks and other resource consumption problems.

An important idea behind the protocol is to use symmetric key encryption to move computation involving public keys from a mobile node to a more powerful node in its domain. Starting from an IPSec-like key exchange protocol in which a mobile node and a correspondent node directly authenticate each other using public-key cryptography, we systematically apply protocol transformations to finally arrive at the protocol which meets the desired design criteria. The protocol transformations are described informally to convey the intuition behind the design. A complete formalization of protocol transformation rules and their correctness will be presented elsewhere.

The second important issue is that of limited authentication infrastructure. While there have been concrete proposals for maintaining a global trust infrastructure, e.g., by extending the Domain Name System (see [6, 7]), successful deployment in the short term seems unlikely. We therefore consider several variants of public key certification : nodes can obtain certificates from a trusted Certification Authority (CA); they can use cached self-signed certificates; or they can obtain certificates online from a nearby server where the server itself might possess either a certificate from a CA or a cached self-signed certificate. As an expository convenience, we present our protocol under the assumption that each authentication server has a public-key certificate issued by a CA. The strong authentication properties discussed above hold under this assumption. We then consider scenarios in which some of the certificates are self-signed. The value of self-signed certificates is that the first time the authentication servers participate in Mobile IP, they exchange self-signed certificates and cache them. In subsequent executions of the protocol, they use the cached self-signed certificates for authentication. While this method is susceptible to a person-in-the-middle attack, an attack is only possible once for each pair of authentication servers (up to the capacity of the certificate cache). An additional advantage of this scheme, which could make a difference during its implementation, is that the structure of the protocol is minimally affected by the available authentication infrastructure - the difference merely lies in the nature of the certificates used and how they are obtained.

The design of security protocols has, in general, been notoriously prone to errors. In the recent past, a fruitful application of formal methods has been in uncovering bugs and proving correctness of a broad class of security protocols. In particular, formal methods have been successfully used to analyze key exchange and authentication protocols [8, 9, 10, 11, 12, 13]. We have formally verified the correctness of our protocol using the finite state analysis tool Murφ [14]. In previous work, Murφ has been used to verify the correctness of several protocols [13, 15, 16, 17]. The fact that the Murφ analysis did not find any bugs, therefore, gives us increased confidence in the correctness of the protocol.

The remainder of this paper is structured as follows. Section 2 describes the requirements for security in Mobile IPv6. Section 3 briefly discusses the previous proposals for authenticating binding updates. In Section 4, we present our basic protocol. Section 5 presents our modelling assumptions and analysis results. In Section 6, we discuss extensions to prevent denial of service attacks. Concluding remarks appear in Section 7.

2. Security Requirements for Address Notification

While address notification makes end-to-end routes shorter, the ability to change routes on the fly introduces a number of security concerns. A Mobile IPv6 address notification message 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 an address notification message from a mobile node (MN), a correspondent node (CN) 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 address notification messages (or binding updates) are described at length in [3]. 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 send false address information if binding updates are not authenticated. In fact, an active attacker can launch a person-in-the-middle attack, pretending to be the default router to the MN and 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 [3], are:

- Identity verification should not rely on the existence of a global PKI.

- Consider the computational capabilities of the MN's and CN's.

- Minimize the number of messages and bytes sent between the participating entities.

- 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 authentication infrastructure. In order for A to directly authenticate herself to 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. In effect, the infrastructure must provide a chain of trust from A to B. 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 4.

3. Previous Proposals

Previous protocols for authenticating binding updates [1, 2, 4, 5] fall short of meeting all the security requirements. We review these protocols to show some of the forms of cryptography that have been considered and to illustrate their shortcomings.

An earlier version of the Mobile IPv6 Internet Draft [2] advocated the use of IPSec and IKE [18] to authenticate binding updates. Although this approach provides strong authentication, it suffers from two serious drawbacks. It requires mobile nodes to possess public key certificates from a global PKI and to perform expensive public-key cryptographic operations.

In contrast, the current Mobile IPv6 Internet Draft [1] offers a solution at the other end of the spectrum. The "return routability procedure" proposed there does not rely on any form of authentication infrastructure. The mobile node sends two cookies to the correspondent node: the HoT cookie and the CoT cookie. The correspondent node returns the HoT cookie to the mobile node's home address and the CoT cookie to its care-of address. In addition it generates two additional cookies of its own - the home cookie and the care-of cookie and sends them to the mobile node. The home cookie is sent with the HoT cookie and the care-of cookie with the CoT cookie. The binding update is cryptographically bound to the cookies generated by the correspondent node. A correspondent node therefore authenticates a mobile node by verifying whether it is reachable at both its home and care-of addresses. However, an active attacker on the path between the mobile node and the correspondent node or between the home agent and the correspondent node can launch a person-in-the-middle attack on this protocol.

Bradner, Mankin and Schiller [4] 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 [5] 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 [19], DIAMETER [20]) 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 [5] 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 cause damage. Also, even if MN and CN detect the attack, the only thing they can do is drop the session, since there is no way to determine which of the exponentials is authentic. So, the attack still succeeds as a denial of service attack.

4. The Protocol

In this section, we propose a specific protocol for authenticating binding updates. Our protocol consists of two phases: (a) an *initialization phase* in which mobile node

(MN) and correspondent node (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). The initialization phase protocol is presented in stages. Starting from an IPSec-like key exchange protocol in which a mobile node and a correspondent node directly authenticate each other using public-key cryptography, we systematically apply protocol transformations to finally arrive at the protocol which meets the design criteria. The protocol transformations aim at moving expensive computations away from mobile nodes as well as reducing the reliance on authentication infrastructure.

Throughout, we assume that each mobile node has a preestablished bidirectional security association with an authentication server in its domain. This is a reasonable assumption and has been recognized as such in [3]. A mobile node's authentication server may be co-located with its home agent or could operate independently. We also assume that the authentication servers are capable of authenticating each other using public key cryptography. Strong authentication is provided if the public-key certificates are issued by a CA; weaker authentication guarantees are offered if some of the certificates are self-signed.

4.1 Notation

The following notation is used in describing the protocol.

MN	Mobile node
CN	Correspondent node
A_{MN}	Authentication server of MN
A_{CN}	Authentication server of CN
BU	Binding update
K_M	Shared secret between MN and A_{MN}
K_C	Shared secret between CN and A_{CN}
K_M^+	Public encryption key of A_{MN}
K_{MN}^+	Public encryption key of MN
$Cert_i$	Certificate of entity i
$Sig_i\{\ldots\}$	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
$MAC(k)\{M\}$	Message auth code of M with key k

4.2 Protocol Description

4.2.1 Initialization Phase

The base two party protocol from which we derive the initialization phase protocol is shown in Figure 1. It is an authenticated Diffie-Hellman key exchange protocol [21, 24] resulting in a shared secret between MN and CN.

$$MN \rightarrow CN : g^x, [Cert_{MN}]$$

$$CN \rightarrow MN : \{g^y\}_{K_{MN}^+}, Sig_{CN}\,(g^y, g^x, MN),$$
$$[Cert_{CN}]$$

$$MN \rightarrow CN : MAC(K_D)\{g^x, CN\}$$

Figure 1. Base two party protocol

The first message is straightforward. Note that it assumes that MN already knows a group and generator that is acceptable to CN. Upon receiving that message, CN generates g^y and computes $K_D = g^{xy}$. She then sends g^y encrypted with $MN's$ public key. This ensures that no attacker can observe g^y. The signature on the second message assures MN that CN did in fact receive g^x, generate g^y and that the message is meant for MN. Omitting $MN's$ identity from inside the signature opens up the protocol to a person-in-the-middle attack similar to the attack on the original Needham-Schroeder protocol [22]. For the purpose of this protocol, $MN's$ identity could be its home address. In the third message, MN sends a keyed message authentication code (e.g., HMAC [25]) of g^x and $CN's$ identity. The key is the Diffie-Hellman secret resulting from the exchange. Since only MN could have recovered g^y and computed K_D, CN can conclude that the message was indeed sent by MN. Including g^x and $CN's$ identity inside the MAC assures CN that MN had sent the original g^x and that CN is the intended recipient of this message.

However, this protocol fails to meet the design requirements on two counts: it requires mobile nodes to be part of a global PKI as well as to perform expensive public-key operations. We now describe a protocol transformation mechanism aimed at overcoming these limitations. The idea is to introduce another protocol principal, A_{MN}, between CN and MN. Instead of sending authenticated data directly to MN, CN sends it to A_{MN}, who in turn forwards it to MN. In effect, the second message of the protocol in Figure 1 gets replaced by two messages - one from CN to A_{MN} and the next from A_{MN} to MN. Public-key cryptography is restricted to the $CN - A_{MN}$ link. Symmetric-key cryptography is used between A_{MN} and MN. The resulting protocol is shown in Figure 2.

The guarantees provided by the second message of the base protocol in Figure 1 are preserved by the second and third message of the protocol in Figure 2. Both these messages (a) preserve the secrecy of g^y; (b) authenticate both g^y and g^x; and (c) include the intended recipient of the message within the authenticated data. Note that the second message includes, within the body of the authenticated data, the component $\{A_{MN}, MN\}$. Along with the intended recipient of the message, this component specifies the path

$CN \rightarrow A_{MN} : \{g^y, g^x, \{A_{MN}, MN\}\}_{K_M^+},$

$Sig_{CN}\{g^y, g^x, \{A_{MN}, MN\}\},$

$[Cert_{CN}]$

$A_{MN} \rightarrow MN : \{g^y, g^x, MN\}_{K_M}$

$MN \rightarrow CN : MAC(K_D)\{g^x, CN\}$

Figure 2. Protocol with one forwarding agent

along which the authenticated data is to be subsequently forwarded. It can be viewed as a generalization of the scenario in the base protocol in which the intended recipient does not need to forward the data to another principal.

In general, a correspondent node (CN) can either be a stationary or a mobile IPv6 node. If it is a stationary node with sufficient computational power, it can very well perform the public-key operations required in the protocol of Figure 2. However, if it possesses limited computational resources or if it does not have a public key certificate whereas some other server in its administrative domain does, then it would be useful to forward the second message of the protocol in Figure 2 through that server to A_{MN}. The resulting protocol is shown in Figure 3.

$MN \rightarrow CN : g^x, [Cert_{A_{MN}}]$

$CN \rightarrow A_{CN} : \{g^y, g^x, \{A_{CN}, A_{MN}, MN\}\}_{K_C},$

$[Cert_{A_{MN}}]$

$A_{CN} \rightarrow A_{MN} : \{g^y, g^x, \{A_{MN}, MN\}\}_{K_M^+},$

$Sig_{A_{CN}}\{g^y, g^x, \{A_{MN}, MN\}\},$

$[Cert_{A_{CN}}]$

$A_{MN} \rightarrow MN : \{g^y, g^x, MN\}_{K_M}$

$MN \rightarrow CN : MAC(K_D)\{g^x, CN\}$

Figure 3. Protocol with two forwarding agents

As before, g^y is sent encrypted throughout and g^y, g^x, and the forwarding paths are included within the body of the authenticated data. Symmetric-key cryptography is used between CN and A_{CN}. Note that whether CN sends the message to A_{MN} directly or through A_{CN} the format of the message is exactly the same. So, based on the available computational power and authentication infrastructure, CN can locally decide which version of the protocol to implement without in any way affecting either MN or A_{MN}.

Public-key Certificates. An issue that merits discussion at this point is the nature of the public-key certificates used and how they are obtained. Two of the protocol principals (CN or A_{CN} and A_{MN}) must possess public-key certificates. Ideally, both of these certificates would be issued by a globally trusted CA or a chain of issuers ending at a CA. In that case, after executing the protocol both MN and CN are guaranteed that they share a secret key with each other. However, in the absence of CA-issued certificates, one or both of the certificates could be self-signed. In that case, the principals will exchange the certificates the first time they execute the protocol and cache them for subsequent use. Self-signed certificates can be created by a node for herself or it could be obained online from a nearby server which owns a certificate (either from a CA or self-signed). How certificates are obtained is thus a matter of local administrative policy. The viability of caching unauthenticated certificates has been tested by SSH and seems to work quite well in practice.

Based on the nature of the certificates, the protocol provides different authentication guarantees. Three possible cases arise: (a) CN (or A_{CN}) uses a self-signed certificate while A_{MN} uses a CA-issued certificate. In that case, upon receiving the last message of the protocol, CN is assured that it has a shared secret with MN. However, MN comes to the same conclusion only under the assumption that the first time the protocol was executed with that CN, a person-in-the-middle attack was not launched. In any case, even if MN sets up a shared secret with a node impersonating CN, authentication is not compromised. Since CN does not know the secret, it will reject binding updates authenticated with it. This results in longer routes through the mobile node's home agent. But the attacker cannot fool CN into accepting a fake binding update by pretending to be MN. (b) A_{MN} has a self-signed certificate while CN (or A_{CN}) has a CA-issued certificate. In that case, the first time CN executes the protocol with MN, it is possible for an attacker to set up a shared secret with CN by pretending to be MN. It can subsequently fool CN into accepting binding updates authenticated with the shared secret. (c) Both A_{MN} and CN (or A_{CN}) have self-signed certificates. As in case (b), an attacker can set up a secret with CN pretending to be MN and then fool her into accepting fake binding updates authenticated with that key. In addition, as in case (a), an attacker can set up a shared key with MN pretending to be CN. However, as discussed above, this does not result in a compromise of the authentication requirement. Indeed for the purpose of authenticating binding updates, it is sufficient if A_{MN} possesses a CA-issued public-key certificate. However, unless CN (or A_{CN}) also possesses a similar certificate, the protocol is open to denial-of-service attacks. This issue is discussed further in Section 6.

A possible instantiation of the concept of self-signed certificates is CAM [26]. When a mobile CAM node is first initialized, she creates a public-private key pair. It next chooses a home address for itself. The routable 64−bit address prefix is obtained by listening for local router advertisements. The lower-order 64 bits are chosen to coincide with a cryptographic one-way hash of the node's public key. Upon receiving a binding update signed with the mobile node's private key, the correspondent node can verify that the mobile node's public key does indeed hash to the lower order bits of her IPv6 address. In our protocol, a similar approach can be adopted for binding $CN's$ or $A'_{CN}s$ public keys to their IP addresses if they are in a position to choose the lower order bits of their IPv6 address.

Minimizing Number of Messages. In this protocol, CN authenticates herself to MN through the chain of trust $CN \rightarrow A_{CN} \rightarrow A_{MN} \rightarrow MN$. 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 [5]. 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 requires at least $n+1$ messages.

Perfect Forward Secrecy. 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 [25]). Note that this variation of the base protocol does not have PFS since the compromise of long term secrets like the shared key between MN and A_{MN} 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.

Other Issues. 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 an authentication server actually owns a 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 authentication server'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 A_{MN} being any wiser. Of course, the use of signatures does provide non-repudiation: if the exchange is recorded A_{MN} can prove later that the adversary claimed ownership of a node that it in fact did not own.

We also assume that the signature scheme is 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 [23] for further discussion on the relative advantages of the two methods).

4.2.2 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 [25]). 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 [2]). 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.

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 briefly outline the general methodology and describe some of the challenges we faced in applying it to this protocol. A general description of the methodology can be found in [13].

5.1 The Murφ Verification System

Murφ [14] is a finite-state verification tool that has been successfully applied to multiprocessor cache coherence protocols and multiprocessor memory models [27, 28]. 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 authentication servers 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.

5.2 The Methodology

We analyzed our protocol 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.* A typical correctness condition would be that the intruder does not learn any secret information. More details about the correctness conditions for our protocol are given in Section 5.4.

4. *Run the protocol* for some specific choice of system size parameters. We have been able to run our protocol with upto 3 mobile nodes and 3 authentication servers, where each mobile node can execute 2 sessions in parallel. Details of execution time appear in Section 5.4.

5.3 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. Murφ is quite useful in discovering authentication bugs since properties like key ownership, source of messages, etc. are easily captured in logical statements. The fact that Murφ did not uncover any bugs in our protocol therefore gives us a fair degree of confidence in its correctness.

5.4 Modelling the Initialization Phase

The Murφ model of the protocol consists of three types of finite state machines corresponding to mobile nodes, authentication servers and intruders. The number of mobile nodes, authentication servers 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 an IP address, the IP address of an authentication server, the secret shared with the authentication server 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 up until 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 authentication server. 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 an authentication server is much simpler. Since an authentication server only forwards messages, her state does not change during the execution of the protocol. The state of an authentication server consists of her certificate, the verification key of the trusted third party (Certification Authority of the PKI) and the shared keys with the mobile nodes in her domain. The transition rules define the sequence of actions that an authentication server 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 [30]), 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 authentication servers 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 authentication servers 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 $(g^x \ mod\ p)^y \ mod\ p = (g^y \ mod\ p)^x \ modp$. (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.

In earlier work [29], Meadows analysed a Diffie-Hellman based key exchange protocol using the NRL Protocol Analyzer. Her framework requires all assumed cryptographic identities to be explicitly specified as rewrite rules. The commutative properties of the Diffie-Hellman algorithm therefore had to be specified as such. Her solution involved developing two sets of operations and rewrite rules, one for initiator and one for responder. In other words, computation of the Diffie-Hellman key was assumed to involve different operations for initiator and responder, even though in fact they are both the same. Two more rewrite rules were then used to reduce the keys computed by the initiator and the responder to the same syntactic expression. The computational hardness property did not have to be explicitly modelled. The very fact that there was no rewrite rule for computing the Diffie-Hellman secret from the public exponentials ensured that it could be done.

We note that our modelling approach is distinct from Meadows'. Since the basic modelling frameworks are quite different, a direct comparison of the relative merits and demerits of the two approaches is not meaningful.

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 [31]. 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 [32]). 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 authentication servers do not create state, memory denial of service attacks are not possible on the authentication servers. Preventing computation denial of service attacks on the authentication servers reduces to the problem of detecting, without performing expensive computations, whether a message has been replayed. Replay attacks on A_{CN} can be prevented by including a sequence number or timestamp inside the encryption in message 2 of the protocol. A_{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 A_{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.

7. Conclusions

In response to the requirement that all location information about a mobile node in IPv6 should be authenticated [2, 3.1], we have proposed a protocol for authenticating binding updates. The computational load on the mobile nodes is minimized, by moving expensive public-key operations to more powerful nodes. The number of messages is kept at a 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. Although, in previous work [29], a Diffie-Hellman based key exchange protocol has been formally analyzed, the framework and modelling technique used there is quite different from ours.

Finally, we have addressed the most difficult adoption issue for authenticated Mobile IPv6: reliance on authentication infrastructure. Our goal has been to make the best use of whatever authentication infrastructure is present, thereby offering a middle path between proposed protocols which require the existence of a global PKI [2] and which do not assume any infrastructure at all [1] and therefore provide weak authentication guarantees. By capturing the differences in authentication infrastructure in the type of public-key certificates, we ensure that the structure of the protocol depends minimally on the available infrastructure. Also, 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. In particular, the exchange of self-signed certificates by authentication servers the first time these servers participate in Mobile IP, allows the protocol to be used without a global PKI.

References

[1] D. Johnson, C. Perkins, Jari Arrko. Mobility Support in IPv6. *Internet Draft*, June 2002.

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

[3] 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.

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

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

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

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

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

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

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

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

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

[13] 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.

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

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

[16] 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.

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

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

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

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

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

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

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

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

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

[26] G. O'Shea, M. Roe . Child-proof Authentication for MIPv6 (CAM). *Computer Communications Review*, April 2001.

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

[28] 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.

[29] C. Meadows. Analysis of the Internet Key Exchange Protocol using the NRL Protocol Analyzer. *In Proc. IEEE Symposium on Security and Privacy*, pages 216-231, 1999.

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

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

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