A METHODOLOGY FOR MECHANICALLY VERIFYING PROTOCOLS USING AN AUTHENTICATION LOGIC*

Munna
Theoretical Computer Science Group
Tata Institute of Fundamental Research
Homi Bhabha Road
Bombay 400005, India
munna@tcd.tifr.res.in

Jim Alves-Foss
Laboratory for Applied Logic
Department of Computer Science
University of Idaho
Moscow, ID 83844-1010 USA
jimaf@cs.uidaho.edu

Abstract

This paper describes a methodology that can be used for rigorously developing authentication protocols for distributed systems. It is based on the logic of authentication proposed by Lampson et al. We implemented the logic of authentication using Higher Order Logic (HOL) as the theorem prover. Based on this implementation, a methodology was developed for analyzing authentication protocols for distributed systems, and was utilized to analyze published authentication protocols. This methodology took into consideration the prudent engineering practices for cryptographic protocol design, proposed by Abadi and Needham. It was observed that formalizing the steps in a protocol let the flaws in the design be easily noticeable. The methodology developed assists in systematically checking for known types of vulnerabilities in authentication protocols.

Keywords: Authentication, Modal Logic, Formal Methods

Introduction

With the ever increasing role of open network communications, such as the Internet it is essential that certain measures be put in place to ensure the security of the systems. One such measure, authentication, enables systems to reliably validate the identity of remote users before granting them access to system resources. This measure is implemented through the use of network authentication protocols.

The development of authentication protocols is difficult, as evidenced by the number of published protocols which have been found to contain flaws [10]. This demonstrates the need for mechanisms that assist protocol designers with the construction and the verification of protocols. To ensure correctness, the security properties of a protocol and the functionality of a protocol are the most important areas to concentrate on while developing a verification method. Formal validation of security mechanisms is a method that can be used to avoid flaws and enhance protocol correctness.

Formal verification of mathematical protocols is usually performed in a theorem proving system. Although *ad hoc* testing techniques may be used, these do not provide the level of assurance required by secure computing systems, since there is the chance of a flaw being overlooked by the test cases used. Hand verification is prone to inconsistencies and is cumbersome. In a formal approach, the specifications are written

*Project sponsored in part by the National Security Agency under Grant Number MDA904-96-1-0108. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation hereon.

in a formal logic such as predicate calculus or higher-order logic. For the work presented here, we used the Higher Order Logic (HOL) theorem proving system [8].

The theory of authentication of Lampson *et al.* [9] addresses different issues concerning authentication in distributed systems. Mechanizing the logic in the theory will lead to mechanizing the specification and verification of authentication protocols. Protocols written in conventional security notation can be translated to the form used in the theory and mechanized directly using a theorem prover like HOL. This will give the authentication protocol designer the ability to mechanically verify the protocol.

Related Work

Relevant related verification work includes the development of a mechanized logic for secure key escrow protocol verification by Schubert and Mocas [14]. They developed a framework to formally specify and verify the correctness of key escrow protocols which were mechanized using HOL. They followed the SVO logic [15] for their work.

Brackin [3] proposed a method to decide cryptographic protocol adequacy with HOL based on the GNY logic [7]. He gives a definitional HOL formalization of the GNY logic for analyzing whether protocols achieve desired communication conditions.

Contributions

The main contributions of the work referred to in this paper to the field of verification of authentication protocols for distributed systems are:

- This is the first report of the logic of authentication of Lampson *et al.* [9] being mechanized in a theorem proving system for verification of protocols for authentication in distributed systems.
- We have mechanized the logic of authentication in HOL such that, given a protocol, it can be converted easily to a form that can be verified automatically. In this process, the assumptions being made become clear so that the occurrences of flaws can easily be spotted, thus making the protocol designer's job much easier.
- The logic that we used has been shown to be implementable in a distributed system [16]. By mechanizing it, we have been able to take the concept of a fault-free protocol for distributed systems closer to reality.

The rest of this paper follows the following format: first we briefly discuss the implementation of the logic. We then focus on using the implementation in a methodical fashion, to demonstrate an approach to systematic analysis of cryptographic protocols. Finally, we discuss uses and future exploration of this material.

Implementation of the Logic of Authentication

The aim of this work was to mechanically implement a modal logic of authentication to verify authentication protocols. The formal model of the authentication logic needed to be developed in a mechanical verification system in order to ensure the accuracy of authentication protocol verification. The mechanical system performs syntax and type checking of the specifications and prevents the proofs from containing logical mistakes. The proofs involved necessitated the use of a verification system which supported higher-order logic and typed lambda calculus. We chose the HOL system for the work because of its support for higher-order logic, generic specifications and polymorphic type constructs. These features directly affect the expressibility of the specification language. In this section we discuss the logic of authentication used in this paper and briefly discuss how it was implemented in the HOL system. The intent of this section is to give

the reader a familiarity of the concepts of the implementation, a full description of the implementation is provided in [11].

Basic Concepts in the Logic of Authentication

In the theory of authentication of Lampson *et al.* [9], the basic conceptual units include *principal* and *statement*. A statement can be a request or an assertion and a principal is the entity that makes a statement. Each type of principal can only **say** (make) statements. Specifically, principals of the type channel can make statements directly, all other principals make statements through a channel. An important concept in this logic is the **speaks for** relation. It lets information be carried forward by incorporating the operation of one principal speaking for another. Lampson *et al.* describe this relation as follows: *A* **speaks for** *B is the fact that if principal A says a sentence s, then this is equivalent to principal B saying s. For example, a channel from a terminal speaks for the user at the terminal.*

The basic operators defined in this logic include **and, quotes, as,** etc. The **and** operator is a conjunctive for principals; a principal can **quote** another; a principal can be in the role of another principal, and it can act **as** that principal.

The HOL System

HOL is a general theorem proving system developed at the University of Cambridge [8, 4] that is based on Church's theory of simple types, or higher–order logic [5]. Similar to predicate logic in allowing quantification over variables, higher–order logic also allows quantification over predicates and functions thus permitting more general systems to be described.

HOL is not a fully automated theorem prover but is more than simply a proof checker, falling somewhere between these two extremes. HOL has several features that contribute to its use as a verification environment: built–in theories, rules of inference for higher–order logic, proof tactics, a proof management system, and a meta-language for extending the prover.

Notations and conventions. Throughout this paper we present several definitions and theorems developed in the HOL system. To make this work more understandable to the reader unfamiliar with HOL syntax, we have run the HOL output through a post-processor. This post-processor transfers HOL special symbols into their logic symbol counterparts. These include:

1. quantifiers $\forall$, $\exists$, λ (forall, there exists, lambda abstraction)
2. logical operators $\neg$, $\wedge$, $\vee$, $\Longrightarrow$ (negation, and, or, implication)
3. comparison operators $>$, $\geq$, $=$, $\leq$, and $<$
4. conditional operator $\rightarrow$ (such that `a` $\rightarrow$ `b` | `c` meaning "if `a` then `b` else `c`")

Variables and terms from the HOL listings are presented in `typewriter` font when discussed in the text of this paper. Definitions in the HOL system are preceded by $\vdash_{def}$. Theorems that have been proven are preceded by $\vdash$.

Implementation of Concepts in Authentication

The first stage in mechanizing the logic of authentication presented in [9] involved the translation of the definitions and conventions of the logic into HOL. Full details of the development of the HOL theory from the authentication logic are provided in [11]. The mappings made in developing the theory included using the HOL type `bool` for *statement* in [9] and creating the new type `principal` to model *principal* in the

authentication logic. The structures of the operators `says, and, quotes`, and `speaksfor` were specified as follows:

```
Constants and Infixes --
  says ":principal -> (bool -> bool)"
  and ":principal -> (principal -> principal)"
  quotes ":principal -> (principal -> principal)"
  speaksfor ":principal -> (principal -> bool)"
```

The properties of these operators were defined in the axioms and theorems, Sn and Pn, according to the corresponding axioms and theorems in Lampson *et al.*'s theory. For example, the theorem S5 corresponding to **S5** in Lampson *et al.*'s theory is:

```
S5
      ⊢ ∀A s s'. A says (s ∧ s') = A says s ∧ A says s'
```

It is equivalent to the original theorem

(S5) $\vdash A$ **says** $(s \wedge s') \equiv (A$ **says** $s) \wedge (A$ **says** $s')$

which states that **says** distributes over $\wedge$. The proofs of the theorems were developed using the HOL theorem prover. Details of all the proofs are given in [11].

The HOL version of the axiom **P1**, that states that $(A \wedge B)$ says something if both A and B say it, is:

```
 P1
      ⊢def ∀A B s. (A and B) says s = A says s ∧ B says s
```

In [11] we show the direct mechanization of all similar basic axioms and theorems involving the principals and the statements in Lampson *et al.* 's [9] theory. In [11] we also illustrate the mechanization of the axioms and theorems involving roles and delegation in the theory. These are also direct implementations of corresponding axioms, theorems and operators in the theory. For further manipulation of the theory and its consequent use, some properties of the operators were proven. Wobber *et al.* [16] describe a few more properties that were built to enhance the original theory. The axioms and theorems corresponding to these properties can also be found in [11]. All of this additional material is beyond the scope of this paper.

Methodology for Verifying Authentication Protocols

As pointed out by Abadi and Needham [2], the use of a set of design principles can reduce errors in the design of cryptographic protocols or make the protocols more efficient. From this work we selected the principles which came within the scope of the theory of authentication and used them to develop a methodology to check for flaws in protocols which allow spoofing and replay attacks. Subsequent subsections provide a detailed illustration of the methodology using examples given in [2].

Methodology

The general steps in mechanizing a protocol and analyzing it in the automated environment are as follows:

1. Translate the protocol into the notation of [9] and then to the HOL notation agreeable to the theory developed. It might be necessary to define new operators and prove corresponding theorems in the theory to incorporate new concepts in the protocol being analyzed.

2. Examine the implications in the protocol and develop rules that the protocol follows to (supposedly) correctly authenticate a channel. For this we analyze each step of the protocol and analyze the preconditions and post-conditions and put them in the form of rules that the protocol follows. While doing this, we can determine the protocol steps that are derived using earlier (independent) steps in the protocol and corresponding rules. Initially we define the set of actual assumptions and requirements of formal verification and then separate the rules for interactions, namely, requests or responses.

3. Develop a model of the protocol which has only the independent steps of the protocol by removing any dependent steps of the protocol, e.g., responses from server.

4. Use the theorem prover to prove that this model of the protocol, along with the rules developed earlier, leads to the authentication of the channel. (Note this only checks that the protocol authenticates a valid user, it does not check if the protocol does not authenticate an invalid user).

5. To check for flaws, we add corresponding assumptions (depending on the flaw that we are trying to find, e.g., vulnerability to spoofing or replay attack) to the protocol model just developed, and check with the theorem prover, whether the protocol incorrectly authenticates the channel. If it does, we have shown that the original protocol is flawed.

6. If, in the previous step, it has been shown that the protocol is flawed, we try to find which part of the protocol has the flaw. This is facilitated by the rules developed from the protocol earlier. Depending on which vulnerability of the protocol that we are checking for, we can try to make changes in the protocol that might eliminate the error. Examples of such changes are discussed by Abadi and Needham in [2].

7. These steps can be repeated to check for the known types of attacks and possible types of errors.

Example of spoofing based on a naming problem in protocol

Abadi and Needham illustrate in [2] that a protocol is prone to spoofing when there is a lack of association of a name (of a principal) with the content of a message. They use the authentication protocol of Woo and Lam [17] based on symmetric key cryptography for this (Example 3.2 in [2]). We use this illustration, along with their possible attack method to demonstrate how to apply the methodology described above to mechanize an authentication protocol to detect a flaw that will allow spoofing in a protocol. Woo and Lam's protocol, as described by Abadi and Needham [2] is as follows:

Message 1	$A \rightarrow B:$	A
Message 2	$B \rightarrow A:$	N_b
Message 3	$A \rightarrow B:$	$\{N_b\}_{K_{as}}$
Message 4	$B \rightarrow S:$	$\{A, \{N_b\}_{Kas}\}_{K_{bs}}$
Message 5	$S \rightarrow B:$	$\{N_b\}_{K_{bs}}$

This is described in the words of Abadi and Needham [2]:

> Here N_b is a nonce, S is a server, and K_{as} and K_{bs} are keys that A and B share with S. Basically, A claims his identity (Message 1); B provides a nonce challenge (Message 2); A returns this challenge encrypted under K_{as} (Message 3); B passes this message on to S for verification, bound with A's name under K_{bs} (Message 4); S decrypts using A's key and re-encrypts under B's (Message 5). If S replies $\{N_b\}_{K_{bs}}$, then B should be convinced that A has responded to the challenge N_b.

Following Step 1 of the methodology we have developed, the protocol can be written in the format recognizable to the theorem prover based on the logic of authentication as:

```
Protocol_3_2
      (∀A B Server Kas Kbs Ch Nb.
      (Kas speaksfor (A and Server) ∧
      Kbs speaksfor (B and Server) ∧
      Ch says (Ch speaksfor A) ∧                              %Message 1%
      Nb isnonce (Ch as A) ∧                                  %Message 2%
      Ch says (Kas says Nb) ∧                                 %Message 3%
      B says (Kbs says (Ch speaksfor A ∧ Kas says Nb)) ∧  %Message 4%
      Kbs says Nb ⟹                                           %Message 5%
      Ch speaksfor A)              %Authentication of channel for A%
```

Note that we include `Kas speaksfor (A and Server)` and `Kbs speaksfor (B and Server)` as they are assumed in the development of the original protocol. Also, we introduce the channel `Ch` as required by the theory of Lampson *et al.* to indicate the channel over which communication occurs. We have also added a new operator `isnonce` as there are no operators in [9] that involve nonces. We use the notation `Nb isnonce (Ch as A)` to indicate that `B` has allocated the nonce `Nb` to the channel `Ch` claiming to be `A`.

Performing Step 2, we analyze the preconditions and post-conditions of each of the steps in the protocol to find dependent messages, or submessages. These are messages that will only be sent if the corresponding preconditions hold. The rules deriving these steps, as well as the rule for the final step of authentication are given as general rules to be used while invoking the protocol. The rules obtained in this case are:

```
Protocol_3_2_Rule1  % Authentication Step %
   ⊢def ∀Server Nb Ch A.
        Server says Nb ∧
        Nb isnonce (Ch as A) ⟹
        Ch speaksfor A

Protocol_3_2_Rule2  % Generate dependent message 5 %
   ⊢def ∀Server A B Kas Kbs Ch Nb.
        B says (Kbs says (Ch speaksfor A ∧ Kas says Nb)) ∧
        Kbs speaksfor (B and Server) ∧
        Kas speaksfor (A and Server) ⟹
        Kbs says Nb

Protocol_3_2_Rule3  % Generate dependent mesage 4 %
   ⊢def ∀s A B Kas Kbs Ch.
        Ch says (Ch speaksfor A) ∧ Ch says (Kas says s) ⟹
        B says (Kbs says (Ch speaksfor A ∧ Kas says s))

private_key_lemma
   ⊢ ∀s K Client Server.
        K speaksfor (Client and Server) ∧
        K says s ⟹
        Server says s
```

In `Protocol_3_2_Rule1`, obtained by analyzing the mechanism involved in the original protocol, if the `Server` says the nonce `Nb` and if it is the nonce originally sent to the channel `Ch` claiming to speak for `A`, then, according to the protocol, `Ch speaksfor A`. This is the final step of the protocol used for validating authentication.

According to Protocol_3_2_Rule2, if Kas and Kbs are private keys of A and B respectively with the server, and B says (Kbs says (Ch speaksfor A ∧ Kas says Nb)), then Kbs says Nb. In other words, when B asks the server to decode the message containing the nonce from A sent on channel Ch, the server decodes it with the key it shares with A. This rule removes dependent message 5.

By Protocol_3_2_Rule3, if Ch says that it speaks for A and and that (Kas says s), then, B will say that (Kbs says (Ch speaksfor A ∧ Kas says s)), according to the original protocol. This rule removes dependent message 4.

The private_key_lemma, also part of the original protocol, states that if K is the shared private key of the pair Client and Server, and if K says s, then we can assume that Server says s.

Following Step 3, we develop a model of the protocol which has only the independent steps of a protocol by eliminating the steps derived with the generic rules of the protocol that were listed above. In this example, we obtain the following:

```
Protocol_3_2
   ⊢ (∀A B Server Kas Kbs Ch Nb.
      (Kas speaksfor (A and Server) ∧
      Kbs speaksfor (B and Server) ∧
      Ch says (Ch speaksfor A) ∧              %Message 1%
      Nb isnonce (Ch as A) ∧                  %Message 2%
      Ch says (Kas says Nb)) ⟹                %Message 3%
      Ch speaksfor A)  %Authentication of channel for A%
```

Step 4 involves the use the theorem prover to prove that this is a valid protocol based on the generic rules extracted earlier. The details of the proof are given in [11].

In Step 5, to check whether this protocol can detect an intruder C impersonating the principal A, while A is off-line, we include the possible messages that C may be using during the protocol run as given by Abadi and Needham [2]:

```
Protocol_3_2_flaw
   ⊢ ∀Nb Nb′ Ch A B C Kbs Kcs Server.
       Kbs speaksfor (B and Server) ∧
       Kcs speaksfor (C and Server) ∧
       Ch says (Ch speaksfor A) ∧
       Ch says (Ch speaksfor C) ∧
       Nb isnonce (Ch as A) ∧
       Nb′ isnonce (Ch as C) ∧
       Ch says (Kcs says Nb) ∧
       Ch says (Kcs says Nb) ⟹
       Ch speaksfor A
```

Here, C first claims to be A and then C, through the channel Ch. B sends nonces Nb and Nb' in response to these two claims. C responds twice with the nonce Nb encrypted with Kcs. B verifies these responses with the server by sending $\{A, \{Nb\}_{K_{cs}}\}_{K_{bs}}$ and $\{C, \{Nb\}_{K_{cs}}\}_{K_{bs}}$. Server responds with $\{Nb''\}_{K_{bs}}$ and $\{Nb\}_{K_{bs}}$ respectively, where $\{Nb''\}_{K_{bs}}$ is the result of decrypting $\{Nb\}_{K_{cs}}$ with K_{as}. Of the two responses of the server, only the second response makes any sense to B, as it has never sent out a nonce Nb''. The response that had a valid meaning was corresponding to the nonce Nb, so it assumes that the channel Ch is speaking for A, since B had assigned the nonce Nb to A. Thus, C could impersonate A, when B is willing to talk to A and C at the same time. When we run this using the theorem prover, we can prove that C can impersonate A, thus letting us detect the flaw.

In Step 6, we try to detect the location of the error in the protocol. As the messages we included were to check whether there is a problem with the association of a name with a message content in a protocol message, the domain of the flaw that needs to be detected gets narrowed down. In the case illustrated, the problem lies with the lack of association of the name with the content of the last message of the protocol (Message 5) which is the message from the server, returning the verified value of the encrypted nonce. This can be observed through the lack of association of the name `A` with the nonce `Nb` in the inference part of `Protocol_3_2_Rule2`. So, we change that part of `Protocol_3_2_Rule2` from `Kbs says Nb` to `Kbs says (Ch speaksfor A and Nb)`. This corresponds to changing the actual protocol message 5 from $\{Nb\}_{K_{bs}}$ to $\{A, Nb\}_{K_{bs}}$, thus fixing the naming problem in the protocol. We can repeat all the steps of the methodology to show that this is true.

Example of replay attack based on use of a compromised key

Abadi and Needham state that the use of a key in the recent past does not guarantee its freshness [2]. They illustrate how leaving this principle out while designing a protocol can cause problems using the Needham-Schroeder protocol (the Kerberos protocol also has similar structure) which is as follows (Example 9.1 of [2]):

Message 1	$A \rightarrow S:$	A, B, N_a
Message 2	$S \rightarrow A:$	$\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
Message 3	$A \rightarrow B:$	$\{K_{ab}, A\}_{K_{bs}}$
Message 4	$B \rightarrow A:$	$\{N_b\}_{K_{ab}}$
Message 5	$A \rightarrow B:$	$\{N_b + 1\}_{K_{ab}}$

When A contacts the server S, A is provided with the session key, K_{ab}, and a certificate encrypted with B's key K_{bs} conveying the session key to B. When B obtains the certificate, to make sure that it got it from A, it carries out a handshake with A with the nonce N_b encrypted with the session key K_{ab}. A sends back $N_b + 1$ encrypted with the same key.

Applying the first step of mechanizing and analyzing this protocol, we build the following:

```
Protocol_9_1
       ∀Ch A B Na Nb Server Kas Kbs Kab.
        Kas speaksfor (A and Server) ∧
        Kbs speaksfor (B and Server) ∧
        Ch says ((A to_communicate_to B) ∧ (Na isnonce Server)) ∧    %Message 1%
        Server says (Kas says ((Na isnonce Server) ∧
                               (Kab speaksfor B) ∧
                               (Kbs says (Kab speaksfor A)))) ∧      %Message 2%
        Ch says (Kbs says (Kab speaksfor A)) ∧                       %Message 3%
        B says (Kab says (Nb isnonce A)) ∧                           %Message 4%
        Ch says (Kab says (nonce_check_function(Nb isnonce A))) ⟹   %Message 5%
        Ch speaksfor A                                 %Authenticating channel for A%
```

Note that the operator `to_communicate_to` and the function `nonce_check_function` were introduced to those already existing in the current theory.

In the second step, we derive the following rules:

```
Protocol_9_1_Rule1  % Authenticate message %
    ⊢def ∀Server Kab A B Nb Ch.
        Server says (Kab speaksfor A) ∧
        B says (Kab says (Nb isnonce A)) ∧
        Ch says (Kab says (nonce_check_function(Nb isnonce A))) ⟹
        Ch speaksfor A

Protocol_9_1_Rule2  % Generate dependent message 2 %
    ⊢def ∀Ch A B Na Server Kas Kbs Kab.
        Ch says (A to_communicate_to B ∧ Na isnonce Server) ⟹
        Server says
        (Kas says
         (Na isnonce Server ∧
          Kab speaksfor B ∧
          Kbs says (Kab speaksfor A)))

Protocol_9_1_Rule3
    ⊢def ∀B Ch Kbs s Server.
        Ch says (Kbs says s) ∧ Kbs speaksfor (B and Server) ⟹
        Server says s
```

Protocol_9_1_Rule1 shows that, if Server announces the session key for communication, and B uses that key to encrypt a nonce and send it to the channel speaking for A and if the channel responds with the result of applying the nonce_check_function on the nonce encrypted with the same session key that B had just used, then, according to the protocol, the channel speaks for A.

According to Protocol_9_1_Rule2, when A requests for a session key to communicate to B and includes a nonce Na in the message, the server responds with the message, signed with the shared key with A, Kas, which has the nonce Na, the session key Kab and a certificate intended for B, which is encrypted with the shared key between Server and B, and conveys the session key Kab to be used to communicate with A.

Protocol_9_1_Rule3 states that if Ch presents the statement s signed with the shared key between Server and B, it implies that the server said s.

The modified version of the protocol developed in the third step is:

```
Protocol_9_1
    ⊢ ∀Ch A B Na Nb Server Kas Kbs Kab.
        Kas speaksfor (A and Server) ∧
        Kbs speaksfor (B and Server) ∧
        Ch says ((A to_communicate_to B) ∧ (Na isnonce Server)) ∧    %Message 1%
        Ch says (Kbs says (Kab speaksfor A)) ∧                      %Message 3%
        B says (Kab says (Nb isnonce A)) ∧                          %Message 4%
        Ch says (Kab says (nonce_check_function(Nb isnonce A))) ⟹   %Message 5%
        Ch speaksfor A                          %Authentication of channel as A%
```

And we prove that this protocol authenticates the channel using the rules illustrated, using the theorem prover in the fourth step. The proof is included in [11].

In the fifth step, for a possible attack method to show that this protocol is not valid, we assume that C tries to establish communication with B after recording all the message exchanges between A and B (messages 3-5) and somehow obtaining the session key K_{ab}. C now replays the message 3 to make B think that A is trying to initiate a new conversation. B then requests a handshake from A by sending A the

message, $\{N'_b\}_{K_{ab}}$. C can intercept this message and return $\{(N'_b + 1)\}_{K_{ab}}$ to B as C has got the key K_{ab}. We check this attack method using the theorem prover and find that the protocol still authenticates the channel in which `C` is speaking, as the channel that is speaking for `A`, without any transmission of Message 1.

```
Protocol_9_1_flaw
    ⊢ ∀Ch A B Na Nb Server Kas Kbs Kab.
        Kas speaksfor (A and Server) ∧
        Kbs speaksfor (B and Server) ∧
        Ch says (Kbs says (Kab speaksfor A)) ∧
        B says (Kab says (Nb′ isnonce A)) ∧
        Ch says (Kab says (nonce_check_function(Nb′ isnonce A))) ⟹
        Ch speaksfor A
```

In the sixth step, to find out where in the protocol the error has occurred, we check the rules used in the proof and notice that `Protocol_9_1_Rule2` was never used in the proof, thus letting `C` use an older session key that the server had issued for `A` to communicate with `B` and the replay attack to make `B` believe that `A` is trying to communicate.

Denning and Sacco [6] point out that this protocol needs a whole new approach to eliminate the error. They suggest the use of time stamps. However, the use of time stamps involves checking the time with the local or synchronized clock to check the time delay between the time stamp and the time of receiving a message, in addition to incorporating the timestamps into the messages. We are not dealing with this in this paper, though this can be done by further developing our scheme for mechanizing the protocols.

Abadi and Needham [2] show that there are ways to increase the efficiency of a protocol by avoiding double encryption and avoiding and use of encryption unnecessarily. We are not exploring methods to develop generic methods meant for improving the efficiency of protocols mechanically. But this is one possible arena where future work can be done, as the principles to be followed to achieve it would be similar to the ones developed for finding faults in protocols.

Conclusions and Further Research

Summary

In this project, we presented the results of implementing a theory of authentication for distributed systems in a formal theorem proving environment. We presented the mechanization of the important sections of the theory of authentication by Lampson *et al.* [9] and the enhancements in [16]. We applied the mechanized model of Lampson *et al.*'s theory to implement the basic protocol units in [1] and other arbitrary authentication protocols. We developed a methodology for analyzing authentication protocols mechanically and applied it to a set of protocols that were illustrated to be faulty in [2] and used it to locate faults in them. It is this mechanism that was presented in this paper, the rest of the results can be found in [11].

Conclusions

The main goal of this work was to show that application of a mechanized logic for authentication can be useful for the design of authentication protocols. In the process, we found that formalizing ideas have some benefits in a protocol verification environment. When the implicit assumptions are stated formally, the flaws that are not easily noticeable become clear.

Using a consistent notation to describe all protocols also helps. It sets the ground for comparing different protocols and standardizing the checking for a given type of flaw in any protocol.

Mechanically verifying protocols takes away the tedium of verification by hand. It enables the verifier to show that the proofs are logically correct, and allows common approaches to verification of protocols to be automated by mechanization. Proof maintenance becomes easy and changes can be implemented easily. Mechanization and application of a different logic for authentication also become easy once the general principles for mechanizing a logic are defined.

References

[1] M. Abadi, M. Burrows, B. Lampson, & G. Plotkin. A calculus for access control in distributed systems. DEC-SRC Technical Report #70, 1991.

[2] M. Abadi & R. Needham. Prudent engineering practice for cryptographic protocols. *Proc. IEEE Symposium on Research in Security and Privacy*, 1994, pp. 122-136.

[3] S. H. Brackin. Deciding cryptographic protocol adequacy with HOL. *Proc. International Workshop on Higher Order Logic Theorem Proving and its Applications*, 1995, pp. 90-105.

[4] A. Camilleri, M. Gordon, and T. Melham. Hardware verification using higher order logic. In D. Borrione (ed), *HDL Descriptions to Guaranteed Correct Circuit Designs*. Elsevier Scientific Publishers, 1987.

[5] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5, 1940.

[6] D. E. Denning & G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533-536, August 1981.

[7] L. Gong, R. Needham & R. Yahalom. Reasoning about belief in cryptographic protocols. *Proc. IEEE Symposium on Research in Security and Privacy*, 1990, pp. 234-248.

[8] M. Gordon. A proof generating system for higher-order logic. Technical Report 103, University of Cambridge Computer Laboratory, January 1987.

[9] B. Lampson, M. Abadi, M. Burrows & E. Wobber. Authentication in distributed systems: Theory & practice. Technical report #83, DEC Systems Research Center, February 1992.

[10] A. Liebl. Authentication in distributed systems: A bibliography. *Operating Systems Review*, 27(4):31-41, October 1993.

[11] Munna. Mechanical Verification of Authentication Protocols for Distributed Systems. MS Thesis, University of Idaho, November 1995.

[12] R. M. Needham & M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993-999, December 1978.

[13] R. M. Needham & M. D. Schroeder. Authentication revisited. *ACM Operating Systems Review*, 21(1):7, January 1987.

[14] T. Schubert & S. Mocas. A mechanized logic for secure key escrow protocol verification. *Proc. International Workshop on Higher Order Logic Theorem Proving and its Applications*, September 1995, pp. 308-323.

[15] P. Syverson & P. van Oorschot. On unifying some cryptographic protocol logics. *Proc. IEEE Symposium on Research in Security and Privacy*, 1994, pp. 14-28.

[16] E. Wobber, M. Abadi, M. Burrows & B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3-32, February 1994.

[17] T. Y. C. Woo & S. S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(1):39-52, January 1992.