

# Requirements for Security Protocols

Wuu Yang

Department of Computer Science  
National Chiao-Tung University  
Hsin-Chu, Taiwan, R.O.C.  
wuuyang@cis.nctu.edu.tw

Rong-Hong Jan

Department of Computer Science  
National Chiao-Tung University  
Hsin-Chu, Taiwan, R.O.C.  
rhjan@cis.nctu.edu.tw

*Abstract:* A correct security protocol should satisfy four requirements: no-intrusion, authenticity, freshness, and secrecy. We divide the four requirements into two groups (one for no-intrusion, authenticity, and freshness and the other for secrecy). For the former we use the *message-exchange forms* and for the latter, we use the *p-knows* and *n-knows* information that is inferred from a latest topological orders of the *trigger-graph*. Based on these methods, flaws and weaknesses hidden in an imperfect security protocol could be uncovered.

*Key-Words:* authentication, authentication protocol, computer communication, computer security, security protocol, verification.

## 1 Introduction

<sup>1</sup> A security protocol is a set of messages that the communicating parties use to establish secure communication channels, mostly by setting up agreed-upon secret keys. Examples of computer security protocols include [10, 11, 12, 16, 8]. However, many security protocols are found to be flawed after they are deployed [1]. Given the rapid growth of computer networks, there is a pressing need for a framework and tools for the development and analysis of security protocols [4]. This implies that protocol verification is a difficult task.

A lot of work [14, 9, 5, 6, 12, 7, 15] was focused on protocol verification. Some researchers propose methods for testing security protocols like testing software [17]. After reviewing the previous work on protocol verification, we feel that the security of a protocol is not a single property. Rather, it is the composition of several more fundamental properties. We list four requirements for an absolutely secure protocols and shows how to enforce the four requirements.

**no-intrusion** No-intrusion means that an attacker cannot intrude into a session of a protocol. If the attacker plans to impersonate a legitimate participant, he has to do it from the very beginning of the session. No-intrusion implies that the attacker cannot inject fake messages in the middle of a session of a security protocol without being detected.

**authenticity** Authentication means that a participant must authenticate other participants in a session before he can trust them. Authentication is usually accom-

plished by demonstrating that a participant knows certain secrets, such as a secret key.

**freshness** Freshness means that messages sent and received in a session are generated specifically for the current session. An attacker cannot use messages from previous sessions in the current session.

**secrecy** Secrecy means that the contents of a message is kept secret between the participants of a protocol session.

The four requirements together can guarantee the *correctness* of a security protocol. No-intrusion assures that the persons sending messages in a session are fixed participants. Their roles in a session will not be stolen by attackers. Authentication means the fixed participants can all show sufficient information to identify themselves. Freshness prevents a participant from mistaking a message from a previous session as a legitimate message for the current session. Secrecy keeps a by-stander from obtaining vital information by eavesdropping on the communication media.

The crux of this paper is that we use two verification processes: First, we make sure that the no-intrusion, authenticity, and freshness requirements are satisfied. We do this by checking that the underlying structure of a security protocol consists of *message-exchange forms* (to be discussed in Section 3). Second, we check that an attacker cannot know the important secrets. We use a *earliest topological order* (to be discussed in Section 4) to infer the information an attack could know immediately before each message of the security protocol is sent.

Clark and Jacob [4] identified seven kinds of common attacks to security protocols. Brackin [3] classified five kinds of security threats: freshness, type, binding, parallel session, and oracle attacks. There are many methods intended for protocol verification. Burrows et al. [2] pro-

<sup>1</sup>The work reported in this paper is partially supported from National Science Council, Taiwan, Republic of China, under grant NSC-94-2213-E-009-029. The authors can be reached by e-mail: wuuyang@cis.nctu.edu.tw and rhjan@cis.nctu.edu.tw.

poses a logic for security protocols. Their logic has much influence in the study verification; however, there are still flaws that escapes their logic. Previously, we proposed an exhaustive testing method for security protocols [17].

The rest of the paper is organized as follows: The next section summarizes the terms and notations used in this paper. The third section discusses message-exchange forms. The fourth section infers knowledge of the attackers as well as the legitimate participants. The last section concludes this paper.

## 2 Preliminaries

In this paper, we will use the following Needham-Schroeder protocol [11, 13] as an example.

1.  $A \rightarrow S : A, B, Na$
2.  $S \rightarrow A : \{Na, B, Kab, \{Kab, A\}_{Kbs}\}_{Kas}$
3.  $A \rightarrow B : \{Kab, A\}_{Kbs}$
4.  $B \rightarrow A : \{Nb\}_{Kab}$
5.  $A \rightarrow B : \{Nb - 1\}_{Kab}$

The key  $Kas$  is a shared key between  $A$  and  $S$ , which is established before a session of the Needham-Schroeder protocol starts. Similarly,  $Kbs$  is a pre-established shared key between  $B$  and  $S$ . After a session of the Needham-Schroeder protocol, the new key  $Kab$  would be known exactly to  $A$ ,  $B$ , and  $S$ .

A *security protocol* consists of an partially ordered sequence of *messages* and a specification. Each message consists of an ordered sequence of *items*. An item could be either un-encrypted (which is called *atomic*) or encrypted. An encrypted item consists of an ordered sequence of subitems and an encryption key.

The specification of a security protocol include (1) the keys and other relevant secrets each participant knows before a session of a security protocol, (2) public information that everybody (including the attacker) knows before a session of a security protocol, and (3) the information each participant and the attacker know and do not know after a session of the security protocol completes.

The legitimate communicating participants have some previously established secrets, such as shared secret keys and public keys. We assume that these previously established keys are well kept away from attackers. A security protocol cannot protect secrets once these keys are lost.

A session of a security protocol is made up of two or more participants, who send and receive messages. The sender of the first message is the *initiator* of the session. Intuitively, the initiator is the participant who intends to communicate with other participants.

The purpose of (a session of) a security protocol is to establish a new key between communicating parties (the initiator of a session and a responder). This new key is called the *jewel* of the session. For example,  $Kab$  is the jewel in the above Needham-Schroeder Protocol. Some security protocols may be capable of establish multiple new

keys among three or or communicating parties. Our technique can be easily extended to these more capable protocols. The intended holders of the jewel are called *propositi*. For instance, in the above Needham-Schroeder Protocol,  $A$  and  $B$  are the propositi whereas  $S$ , who knows the jewel, is not.

## 3 How can a receiver $R$ believe in a message that he receives?

For a receiver to believe in the authenticity and freshness of a piece of information, the information must be properly enclosed in messages. The messages of a security protocol must consist of *message-exchange forms*. (We can guarantee the secrecy of a piece of information only by examining the whole security protocol and inferring what an attacker can know. We will study secrecy guarantee in the next section.)

The message-exchange form is either a basic form or its variations. The basic form (from the perspective of the receiver  $R$ ), shown below, consists of a pair of messages: a challenge and a response. The challenge need not be encrypted; however, the response is usually encrypted.

$$\begin{array}{ll} R \rightarrow \dots : tokenr & \text{--- a challenge} \\ \dots \rightarrow R : \{jewel, tokenr\}_{Kp} & \text{--- a reply} \end{array}$$

Here  $tokenr$  is generated by the receiver  $R$  and  $Kp$  is the key that is agreed upon between  $R$  and (1) the intended producer of the jewel or (2) a participant who has already believed in *jewel*.  $Kp$  could be a shared that is established before the current session of the security protocol starts or it could be a key that is established in previous steps of the current session.

These two messages guarantee, to  $R$ , the freshness and authenticity of the jewel *under the provision that the attacker does not know jewel and  $Kp$  and the receiver  $R$  must be able to know  $Kp$* . (There is a provision in every form discussed in this section. In the next section, we will show a method to check the provisions.)

To  $R$ , the freshness of *jewel* is guaranteed by  $tokenr$ , which is generated by  $R$ . The authenticity of *jewel* is guaranteed by  $Kp$ , which is assumed to be a well-kept secret between  $R$  and the legitimate participant who creates the encrypted item  $\{jewel, tokenr\}_{Kp}$ . Note that the two messages guarantees the authenticity and freshness of *jewel* *only to  $R$* , but not to other participants of the security protocol. Usually,  $R$  must present a proof of his identity additionally.

$tokenr$  can be viewed as a *passport* announced by  $R$ . Then  $R$  will treat whatever is bound with *token* (by a suitable key) as packaged later than  $R$  announced  $tokenr$ . Freshness is therefore guaranteed.

A slight variation of the second message is  $\{jewel, \{f(tokenr)\}_{K, \dots}\}_{Kp}$ , where  $f$  is a fixed function, such as one-way hash, increment, or decrement, that

the receiver can verify and  $K$  is a key known to  $R$ . In this variation, either  $jewel$  or  $tokenr$  can be encrypted in order to achieve secrecy. The use of the function  $f$  is to avoid encrypting the same item (in the case,  $tokenr$ ) several times. Certain encryption algorithms become easier to break if the same item is encrypted several times.

The whole second message can also be encrypted several times, such as  $\{\{\{jewel, tokenr\}_{K_p}\}_{K_1}\}_{K_2}\}_{K_3}$ , as long as the receiver can decode it.

*Example.* Assume  $R$  and  $P$  share a previously established secret key  $K_{rp}$ . Consider the following three protocols:

Protocol 1:

$R \rightarrow P : tokenr$

$P \rightarrow R : \{jewel, tokenr\}_{K_{rp}}$

In Protocol 1,  $R$  generates a new token  $tokenr$  and sends it to  $P$ .  $P$  sends back a new key  $jewel$ , which is generated by  $P$ . Protocol 1 establishes a fresh and authenticated key  $jewel$  between  $R$  and  $P$

Protocol 2:

$R \rightarrow P : \{tokenr, K_{new}\}_{K_{rp}}$

$P \rightarrow R : \{jewel, tokenr - 1\}_{K_{new}}$

Protocol 2 achieves a similar effect as Protocol 1.

Protocol 3:

$R \rightarrow P : \{tokenr, jewel\}_{K_{rp}}$

$P \rightarrow R : \{tokenr + 1\}_{jewel}$

Protocol 3 is flawed in that an attacker can pretend to be  $R$  and re-sends a previous message  $\{tokenr, jewel\}_{K_{rp}}$ .  $P$  will believe in the authenticity (but not the freshness) of  $jewel$  if this protocol can guarantee that only  $P$  and  $R$  knows the key  $K_{rp}$ . However, as far as  $R$  is concerned,  $jewel$  is fresh and authenticated (since it is  $R$  who generates  $jewel$ ).

An alternative, more flexible message-exchange form is shown below:

$R \rightarrow \dots : tokenr$

$\dots \rightarrow R : \{tokenp, tokenr\}_{K_p}$

$\dots \rightarrow R : \{jewel, tokenp\}_{K_p}$

This alternative form can be viewed as two runs of the basic form.  $tokenr$  guarantees the freshness of  $tokenp$ , which in turn guarantees the freshness of  $jewel$  under the provision that the attacker does not know  $jewel$  and  $K_p$ .  $tokenp$  must be generated by the party  $P$ .

The token  $tokenp$  may be considered as an intermediate jewel. It can be used to guarantee the authenticity and freshness of other jewels. For instance, we may also use  $tokenp$  to encrypt the jewel. The third message could be replaced by the following message:

$\dots \rightarrow R : \{jewel\}_{tokenp}$

The provision for this variation is that the attacker does not know  $jewel$ ,  $K_p$ , and  $tokenp$  since  $tokenp$  is used to encrypt other items.

We can generalize the above form further.

$R \rightarrow \dots : tokenr$

$\dots \rightarrow R : \{tokena, tokenr\}_{K_a}$

$\dots \rightarrow R : \{tokenb, tokena\}_{K_b}$

$\dots \rightarrow R : \{tokenp, tokenb\}_{K_p}$

$\dots \rightarrow R : \{jewel, tokenp\}_{K_p}$

The provision here is that the attacker does not know  $K_a$ ,  $K_b$ ,  $K_p$ , and  $jewel$ .

*Example.* Consider the following protocol:

Protocol 4:

1.  $A \rightarrow B : A, nonceA$

2.  $B \rightarrow S : A, nonceA, B, nonceB$

3.  $S \rightarrow B : \{Kab, nonceB\}_{K_{bs}}$

4.  $S \rightarrow A : \{Kab, nonceA\}_{K_{as}}$

We claim that a security protocol is correct if it consists of message-exchange forms for the propositi (and the provisions in the forms are observed) for each legitimate participant.

For Protocol 4, there are two message-exchange forms, one for each propositi ( $A$  and  $B$ ):

message-exchange form for $A$
1. $A \rightarrow \dots : \dots nonceA$
4. $\dots \rightarrow A : \{Kab, nonceA\}_{K_{as}}$
message-exchange form for $B$
2. $B \rightarrow \dots : \dots nonceB$
3. $\dots \rightarrow B : \{Kab, nonceB\}_{K_{bs}}$

There is a freshness attack on the Needham-Schroeder protocol [11, 4]. Suppose the secret key  $Kab$  in the first session is compromised and is known to the attacker. The attacker can re-send the third message of the first session as the third message of the second session. Then  $B$  is fooled to take the old key  $Kab$  as the new secret key for the second session. The real trouble lies in that, though  $B$  can decrypt the third message,  $B$  does not have the guarantee that the message is fresh.

Our technique clearly shows that the message-exchange form for the propositus  $B$  is missing. There is only one message-exchange form, which is intended for  $A$ .

message-exchange form for $A$
1. $A \rightarrow \dots : \dots Na$
2. $\dots \rightarrow A : \{Na, \dots Kab, \dots\}_{K_{as}}$
message-exchange form for $B$
missing

In this section, we make an implicit assumption: The legitimate parties of the protocol— $A$ ,  $B$ , and  $P$ —who possess the keys  $K_a$ ,  $K_b$ , and  $K_p$ , respectively, are all well-behaved. When an attacker tries to fool others, he never uses his real identity (so as to avoid being caught in the future). The attacker always impersonates as someone else. This is not a severe limitation. Most security protocols also make such an assumption.

## 4 What does an attacker know?

The information contained in the messages of a security protocol may belong to one of the two categories:

**a. transient information** those that is used only in the current session

**b. sustained information** those that may be used in subsequent sessions.

*Example.* Consider the following protocol:

$A \rightarrow B : \{K_{session}, K_{new}\}_{K_{old}}$

In this protocol,  $K_{session}$  is the key for the current session and  $K_{new}$  will replace  $K_{old}$  in the next session. Thus,  $K_{session}$  is a piece of transient information while  $K_{new}$  is sustained. (In this case,  $K_{session}$  is the jewel while  $K_{new}$  is just a piece of information transmitted in the current session.)

Since sustained information gathered from one session might be useful in subsequent sessions, an attacker may quietly observe several sessions of the security protocol to obtain sufficient sustained information before launching his attack. Thus, we first infer what an attacker can know after he observes enough sessions and then we infer what the attacker can know immediately before each message of the attacked session is sent out.

#### 4.1 What an attacker can know before he launches an attack

We assume that the attacker does not know any secrets initially. The legitimate parties know suitable keys before a session of the security protocol starts. We also assume that the transmission media is subject to eavesdropping. This implies that an attacker is able to collect all messages, though he probably could not decrypt the encrypted items. It is easy to infer what the attacker could know at the end of the whole session. There are two rules concerning the inference:

1. If an item is transmitted in plain text, the attacker is assumed to know the item right away.
2. If the attacker knows a key  $K$  and items encrypted with  $K$ , then the attacker can decrypt the encrypted items. This rule is based on the generally accepted assumption that the encryption algorithm itself is known to the public.

*Example.* Consider the following protocol:

$A \rightarrow B : K$

$B \rightarrow C : \{Q\}_K$

$C \rightarrow D : \{R, S\}_Q, \{T\}_{K_{cd}}$

At the end of a session of the protocol, the attacker would know  $K$ ,  $\{Q\}_K$ ,  $Q$ ,  $\{R, S\}_Q$ ,  $R$ ,  $S$ , and  $\{T\}_{K_{cd}}$ , but he knows neither  $K_{cd}$  and  $T$ .

#### 4.2 What the attacker can know immediately before a message is sent

A common, but implicit property of a security protocol is that there is an *initiator* (who is usually the sender of the first message). When a participant receives a message, he will send another message to another participant. We say the first message *triggers* the second. We may draw

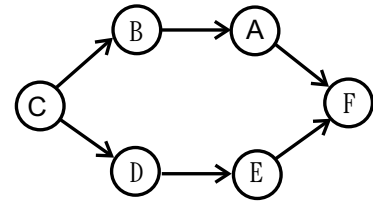


Figure 1. A directed acyclic graph

a trigger-graph in which each vertex represents a message. If message  $A$  triggers another message  $B$ , there will be an edge  $A \rightarrow B$ . Usually the trigger-graph is a linear graph. However, sometimes a message may trigger two or more messages. In Protocol 4 in Section 3, the second message triggers the third and the fourth messages. In that case the trigger-graph is an acyclic directed graph.

On a trigger-graph, we may derive topological orders of the vertices. Among the topological orders, we wish to find the *latest topological order*.

*Definition.* Let  $G$  be a directed acyclic graph  $G$ . A *topological order* is a linear order of the vertices of  $G$  such that if there is an edge  $X \rightarrow Y$  in  $G$ , then  $X$  appears before  $Y$  in the linear order.

Note that a directed acyclic graph may have more than one topological order. In a topological order, the first vertex is assigned the *sequence number* 1; the second is assigned 2. Similarly for other vertices.

*Definition.* Let  $A$  be a vertex in a directed acyclic graph  $G$ , a *latest topological order with respect to A* is a topological order in which the sequence number of  $A$  is no smaller than that in any other topological order.

*Example.* Consider the graph in Figure 1. The orders  $[C, D, B, E, A, F]$  and  $[C, D, E, B, A, F]$  are latest topological orders with respect to vertex  $A$  because in both orders, the sequence number of  $A$  is 5. However, the topological order  $[C, B, D, A, E, F]$  is not a latest topological order with respect to  $A$  because the sequence number of  $A$  is 4.

In terms of the trigger-graph, the latest topological order with respect to a vertex (i.e., a message)  $A$  indicates the message  $A$  is delayed as much as possible. From the latest topological order with respect to vertex  $A$ , we may infer what an attacker can possibly know immediately before message  $A$  is sent out.

It is quite easy to compute a latest topological order with respect to a vertex  $A$  in a directed acyclic graph. We partition the set of vertices into two categories: those that are not reachable from  $A$  and those that are. The vertex  $A$  will be put in the former category. Then the concatenation of topological orders of the two categories is the required latest topological order.

### 4.3 Two kinds of knowledge

After determining a proper order of message transmission, we will infer the knowledge of the legitimate participants and possibly the attacker.

In order to keep certain items secret, we encrypt them with keys. These encryption keys in turn need to be kept secret. We may use other keys to encrypt these keys. The same argument applies to these other keys. Eventually, we use pre-established encryption keys, that is, keys that are established before a session of a security protocol.

In general, for a deeply encrypted item such as  $\{X, \{Y, \{Z\}_{K3}\}_{K2}\}_{K1}$ , at least one of  $K1$ ,  $K2$ , and  $K3$  must be kept secret from each person who is not supposed to know the item  $Z$ .

*Definition.* A participant who is entitled to know the jewel is called a *knowledgeable*.

The propositi are always knowledgablees whereas an attacker is never a knowledgeable. Other participants may also be knowledgablees. For instance, in the Needham-Schroeder protocol,  $A$ ,  $B$ , and  $S$  are all knowledgeablees. However, an attacker is not

It is the designer of a protocol who shall specify explicitly the knowledgeablees in his protocol. The responsibility of a protocol analyzer is to verify the designer's specification.

In order to check the specification of who knows what in a security protocol, there are two cases to consider:

**Positive Side** We need to guarantee all the knowledgeablees are informed the jewel in some form that he could understand, such as in an encrypted form that he could decrypt. That is, *every knowledgeable p-knows the jewel.* (We will define *p-knows* later.)

**Negative Side** We need to guarantee nobody except the knowledgeablees is able to know the jewel. That is, *no one except the knowledgeablees n-knows the jewel.* (We will define *n-knows* later.)

The main difference between *p-knows* and *n-knows* lies in, for *p-knows*, a participant can examine only messages sent to him whereas, for *n-knows*, an attacker can examine all messages sent over the transmission medium.

#### 4.3.1 The positive side

Let  $Z$  be an item. Let  $E$  be a participant. For every occurrence of  $Z$  in the message sent to  $E$ , say  $\{X, \{Y, \{Z\}_{K3}\}_{K2}\}_{K1}$ , a sequence of keys that are used to encrypt  $Z$ , in this case,  $[K1, K2, K3]$ , is created. If  $Z$  appears as a plain (that is, un-encrypted) item in a message sent to  $E$ , an empty sequence is created. The set of all such sequences of keys is denoted as  $\Sigma_{Z,E}$ .

*Definition.* We say that  $E$  *p-knows*  $Z$  if and only if (1) there is at least one sequence of keys in  $\Sigma_{Z,E}$ , say  $[K1, K2, \dots, Km]$ , such that  $E$  p-knows every  $Ki$  in this

sequence; (2)  $E$  knows  $Z$  before the session starts (for instance, Alice p-knows the pre-established key  $Kab$  shared between her and Beth); or (3)  $E$  generates  $Z$ .

It is straightforward to construct the set  $\Sigma_{Z,E}$ , for every  $Z$  and  $E$ —simply by examining every message in the protocol. We can use an iterative algorithm to determine if  $E$  p-knows  $Z$ , for every participant  $E$  and item  $Z$ .

**Theorem.** If a knowledgeable p-knows the jewel, then the knowledgeable eventually knows the jewel.

This theorem implies that the protocol successfully informs every knowledgeable the jewel.

#### 4.3.2 The negative side

Let  $Z$  be an item. For every occurrence of  $Z$  in any message, say  $\{X, \{Y, \{Z\}_{K3}\}_{K2}\}_{K1}$ , a sequence of keys that are used to encrypt  $Z$ , in this case,  $[K1, K2, K3]$ , is created. If  $Z$  appears as a plain (that is, un-encrypted) item in a message, an empty sequence is created. The set of all such sequences of keys is denoted  $\Omega_Z$ .

*Definition.* Let  $E$  be a participant or an attacker. We say that  $E$  *n-knows*  $Z$  if and only if there is at least one sequence of keys in  $\Omega_Z$ , say  $[K1, K2, \dots, Km]$ , such that  $E$  p-knows or n-knows every  $Ki$  in this sequence.

It is also straightforward to construct the set  $\Omega_Z$ , for every  $Z$ —simply by examining every message in the protocol. We can use an iterative algorithm to determine if  $E$  p-knows  $Z$ , for every participant  $E$  and item  $Z$ .

**Theorem.** If no one except the knowledgeablees n-knows the jewel, then no one except the knowledgeablees knows the jewel.

This theorem implies that the protocol successfully hides the jewel from everyone except the knowledgeablees.

*Example.* We use two tables to summarize the *p-knows* and *n-knows* relations for the Needham-Schroeder protocol. Here is the *p-knows* relation:

<i>p-knows</i>	pre-established atomic items	inferred atomic items
$S$	$Kas, Kbs$	$Na, Kab$
$A$	$Kas$	$Na, Kab, Nb$
$B$	$Kbs$	$Kab, Nb$

The *p-knows* table is good for checking if a security protocol successfully distribute the relevant keys to the intended participants. Usually, this is not a problem. The major problem in analyzing a security protocol is to decide what an attacker could know. This problem can be analyzed with the help of the *n-knows* table, shown below.

<i>n-knows</i>	pre-established atomic items	inferred atomic items
$S$	$Kas, Kbs$	$Na, Kab, Nb$
$A$	$Kas$	$Na, Kab, Nb$
$B$	$Kbs$	$Kab, Nb, Na$
$D$ (a by-stander)		$Na$

The *n-knows* table is more useful in estimating what attacker could know if he is one of the participants of the protocol or if he is just a by-stander (by passively eavesdropping on the communication lines).

## 5 Conclusion

We have proposed four requirements of a *correct* security protocol—no-intrusion, authenticity, freshness, and secrecy. The first three can be guaranteed with message-exchange forms while the fourth can be guaranteed with *p-knows* and *n-knows*, which are inferred from the latest topological orders of the trigger-graph.

## References

- [1] M. Abadi and R. Needham, "Prudent engineering practice for cryptographic protocols," *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, pp. 6-15, January 1996.
- [2] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Trans. Computer Systems*, Vol. 8, No. 1, pp. 18-36, 1990.
- [3] S.H. Brackin, "Automatically detecting most vulnerabilities in cryptographic protocols," *Proc DARPA Information Survivability Conference and Exposition 2000 (DISCEX 00)*, Vol. 1, pp. 222-236, January 25-27, 2000.
- [4] J. Clark and J. Jacob, "A survey of authentication protocol literature, Version 1.0," technical report, York Univ., November 1997, available at <http://www-users.cs.york.ac.uk/jac/>.
- [5] J. Heather, G. Lowe, and S. Schneider, "How to prevent type flaw attacks on security protocols," In *Proceedings of 13th IEEE Computer Security Foundations Workshop*, 255-268, 2000.
- [6] P. Janson, G. Tsudik and M. Yung, "Scalability and flexibility in authentication services: The KryptoKnight Approach," *Proceedings of INFOCOM'97*, Vol. 2, pp. 725 -736, 1997.
- [7] A. Kehne, J. Schonwalder, and H. Langendorfer, "A nonce-based protocol for multiple authentication," *AMC Operating Systems Review*, Vol. 26, No. 4, pp. 84-89, October 1992.
- [8] C. Laferriere and R. Charland, "Authentication and authorization techniques in distributed systems," In *Proceedings of IEEE 1993 International Carnahan Conference on Security Technology*, pp. 164-170, 1993.
- [9] Y. Li, W. Yang, and C.W. Huang, "Preventing type flaw attacks on security protocols with a simplified tagging scheme," *Journal of Information Science and Engineering* (accepted), July 2004.
- [10] G. Lowe, "A hierarchy of authentication specifications," In *Proceedings of 10th Computer Security Foundations Workshop*, pp. 31-43, June 1997.
- [11] R.M. Needham and M.D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of ACM*, Vol. 21, No. 12, pp. 993-999, December 1978.
- [12] B.C. Neumann and S.G. Stubblebine, "A note on the use of the timestamps as nonces," *ACM Operating Systems Review*, Vol. 27, No. 2, pp. 10-14, April 1993.
- [13] B.C. Neumann and Theodore Ts'o, "Kerberos: An authentication service for computer network," *IEEE Communications Magazine*, Vol. 32, No. 9, pp. 33-38, September 1994.
- [14] L.C. Paulson, "Proving security protocols correct." In *Proceedings of 14th Symposium on Logic in Computer Science*, 370-381, 1999.
- [15] J.G. Steiner, C. Neumann, and J.I. Schiller, "An authentication service for open network systems," In *Proceedings of the Winter 1988 USENIX Conference*, pp. 191-202, 1988.
- [16] T.Y.C. Woo and S.S. Lam, "A lesson on authentication protocol design," *ACM Operating Systems Reviews*, 28(3), 24-37, 1994.
- [17] W. Yang, "Uncovering Attacks On Security Protocols," In *Proc. International Conf. Information Technology and Applications* (Sydney, Australia), Vol. 2, 572-575, July 4-7, 2005.