

# A logic approach to the verification and testing of security protocols

WUU YANG

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

CHEY-WOEI TSAY

Computer Science and Information Management Department  
Providence University  
Taichung County, Taiwan  
e-mail: cwtsay@pu.edu.tw

## ABSTRACT

Security is an ever important issue in computer networks. Current network security, which is mostly based on (symmetric and asymmetric) key encryption, depends on the soundness of the security protocols as well as the strengths of the encryption algorithms. We propose a logic framework to verify or test the soundness of the security protocols. The logic framework can identify the hidden weaknesses in a protocol. When a verified protocol *does* break down, this logic framework can also quickly point out the guilty assumptions made in the framework that are responsible for the breakdown of the protocol security. The information is useful in mending the broken protocol.

## KEY WORDS

logic, network security, security protocol, protocol testing, protocol verification

## 1 Introduction

With the fast advances in the technologies and applications in computers and computer networks, security will become a more and more important issue in the coming years.

Current encryption-based network security depends on two factors: the strengths of the (symmetric and asymmetric) encryption algorithms and the soundness of the security protocols that make use of the encryption algorithms. Our study is concerned with the second factor.

*Security protocols* in this paper means the process of exchanging, validating, and updating the various encryption and decryption keys among the communication partners. The security protocols are first studied in [1], which leads to the Kerberos system [2]. There are various security protocols such as the authentication protocols [3, 4, 5, 6, 7, 8] and the public-key protocols [9]. As Burrow et al. point out, most of the current security protocols contain redundancies or security flaws [10].

There are two types of security protocols: the static ones and the dynamic ones. The main difference is that the dynamic protocols may generate new keys and revoke existing keys as time goes by. The static protocols allow logically complete verification while the dynamic protocols can only be selectively tested.

Verification and testing of a security protocol is an

important ingredient of the overall system security. Verification and testing can not only indicate the hidden defects in the protocol, but they can also point out the potential weaknesses of the protocol.

We adopt a logic approach to the verification and testing problem. Assumptions, properties and requirements of a protocol are expressed in terms of logic formulae. Inference is used to show the consistency or inconsistency among the formulae.

The approach presented in this paper could serve as a new tool in the toolkit of protocol designers. Though practical protocols are more complex than the examples shown here, a practical protocol can sometimes be viewed as the composition of simpler protocols, some of which could be verified or tested with our approach.

In this paper we use a few standard notations for logic. Introduction to these notations can be found in most textbooks on logic, such as [11], [12], and [13].

Our technique share a similar target as that of Burrows et al. [10]. They differ in that Burrows et al.'s work focused on the verification of the authentication process while ours aims at the detection of leakage of secrets. Burrows et al.'s work makes use of Hoare logic [14] to annotate beliefs and facts in each step of a protocol. Their results are similar to ours in that both their logic and our technique cannot prove the security of protocols in general. With our technique, the protocol designer needs to invent a scenario that shows the existence of a fault in a protocol. The same is true with Burrows et al.'s technique [10].

The remainder of this paper is organized as follows: the next section presents our technique in terms of a simple security protocol. The third section describes a security protocol based on basic asymmetric cryptography. The fourth section illustrates that our technique is able to identify a defect in a defective protocol. The last section summarizes our approach.

## 2 A simple protocol

We first consider a very simple protocol. Assume that there are three roles in the exchange of messages: the sender, the receiver, and the bystander. The sender and the receiver necessarily need to know the message in this protocol. Thus, only the bystander is assumed to be the attacker.

(In other more complicated protocols, the sender and/or the receiver may not know the *contents* of the message. We do not consider that case in this section.)

In the beginning, the sender and the receiver share a master key, which the bystander does not know. When the sender and the receiver want to exchange messages securely, the sender first generates a session key. The sender encrypts the session key with the master key and sends the encrypted session key to the receiver. The receiver, knowing the master key, can obtain the session key by decryption. Later, when the sender and the receiver want to exchange messages, one first encrypts the message with the session key and send the encrypted message to the other, who, knowing the session key, may decrypt the encrypted message. The bystander will be unable to know the master key, the session key, and the messages, assuming that both the sender and the receiver are loyal to each other and the encryption/decryption algorithms are un-breakable. (The session key might expire after a period of time. However, we do not take into account the process of key expiration in this section.)

This protocol is very simple and it would be easy to verify the security of the protocol.

For this protocol, we may list the following axioms: ( $S$ ,  $R$ ,  $B$ ,  $Master$ , and  $Session$  denote the sender, the receiver, the bystander, the master key, and the session key, respectively. The notation  $\{Message\}_{Session}$  means that the message is encrypted with the session key.)

**A1**  $knows(S, Master)$ .

**A2**  $knows(R, Master)$ .

**A3**  $knows(S, Session)$ .

**A4**  $\neg knows(B, Master)$ .

**A5**  $public(\{Session\}_{Master})$ .

**A6**  $public(\{Message\}_{Session})$ .

Axioms A1 through A4 states the initial knowledge of the three roles. Axioms A5 and A6 implies that the communication channel is insecure. That is, every packet transmitted over the channel is open to the public.

We also need a few axioms to define the predicates and the encryption behaviors. The axiom B1 states that all messages sent through the insecure channel is open to the public. Note that the variable  $m$  in Axiom B1 denotes a key, a plain message, or an encrypted message. The axiom B2 states that anybody with a key can decrypt all the messages encrypted with that key. The axioms B3 and B4 state that a person cannot know the decrypted message unless he knows the encrypted message and has the decryption key. Axioms B3 and B4 may seem redundant. They actually reflect our assumption of the surrounding environment and the assumption that the encryption algorithm is absolutely unbreakable. Though there is always a slim chance

of breaking the encryption algorithm however strong the algorithm might be, our assumption is reasonable in that we attempt to discover the weaknesses of the security protocols, not that of the encryption algorithms, in this study.

**B1**  $\forall(x)\forall(m)[public(m) \rightarrow knows(x, m)]$ .

**B2**  $\forall(x)\forall(m)\forall(k)[knows(x, \{m\}_k) \wedge knows(x, k) \rightarrow knows(x, m)]$ .

**B3**  $\forall(x)\forall(m)\forall(k)[knows(x, \{m\}_k) \wedge \neg knows(x, k) \rightarrow \neg knows(x, m)]$ .

**B4**  $\forall(x)\forall(m)\forall(k)[\neg knows(x, \{m\}_k) \rightarrow \neg knows(x, m)]$ .

From Axioms A1 through A6 and B1 through B4, we want to show that both the sender and the receiver are able to read the messages, but the bystander cannot. We also need to show that the axiomatic system is consistent. If not, anybody can read any message. These requirements are formulated as the following items C1 through C4

**C1**  $knows(S, message)$ .

**C2**  $knows(R, message)$ .

**C3**  $\neg knows(B, message)$ .

**C4** The formulae in A1-A6 and B1-B4 are consistent.

In case neither  $knows(B, message)$  nor  $\neg knows(B, message)$  may be induced from A1 through A6 and B1 through B4, additional axioms (that is, assumptions) would be needed.

### 3 Secure communication based on public-key encryption

Next we will consider a slightly more complicated example—secure communication based on the public-key encryption [9]. We assume that the receiver has a pair of keys. He keeps the private key to himself but advertises the public key. The sender first generates a session key, encrypts it with the receiver's public key, and sends the encrypted session key to the receiver. When the sender wants to send a message to the receiver, he encrypts the message with the session key and sent the encrypted message over the communication channel.

For this protocol, we may list the following axioms: ( $RPrivate$  and  $RPublic$  denotes the private and public keys of the receiver, respectively.)

**A1**  $pair(RPrivate, RPublic)$ .

**A2**  $pair(Session, Session)$ .

**A3**  $knows(R, RPrivate)$ .

**A4**  $public(RPublic)$ .

**A5**  $knows(S, Session)$ .

**A6**  $\neg knows(S, RPrivate)$ .

**A7**  $\neg knows(B, RPublic)$ .

**A8**  $public(\{Session\}_{RPublic})$ .

**A9**  $public(\{Message\}_{Session})$ .

**A10**  $knows(S, Message)$ .

Note that in this protocol, we actually used two encryption algorithms: a symmetric algorithm using the key  $Session$  and an asymmetric one with the key pair  $RPrivate$  and  $RPublic$ . However, for our purpose, the symmetric encryption algorithm can be considered as an asymmetric one in which the encryption key serves as both the private key and the public key. Axioms A1, A2, A8 and A9 above are based on this simplification.

We also need a few axioms to define the predicates  $public$  and  $pair$  and the encryption and decryption behaviors.

**B1**  $\forall(x)\forall(m)[public(m) \rightarrow knows(x, m)]$ .

**B2**  $\forall(x)\forall(m)\forall(j)\forall(k)[knows(x, \{m\}_k) \wedge knows(x, j) \wedge pair(j, k) \rightarrow knows(x, m)]$ .

**B3**  $\forall(x)\forall(m)\forall(k)[knows(x, \{m\}_k) \wedge \neg knows(x, j) \wedge pair(j, k) \rightarrow \neg knows(x, m)]$ .

**B4**  $\forall(x)\forall(m)\forall(k)[\neg knows(x, \{m\}_k) \rightarrow \neg knows(x, m)]$ .

**B5**  $\forall(j)\forall(k)[pair(j, k) \rightarrow pair(k, j)]$ .

**B6**  $\forall(i)\forall(j)\forall(k)[pair(i, j) \wedge pair(i, k) \rightarrow j = k]$ .

Axiom B2 states the asymmetric decryption algorithm. Axioms B3 and B4 states the assumption that a known, encrypted message can be decoded only by the key that, together with the encryption key, forms a key pair. Axioms B5 and B6 describe the properties of the encryption algorithm.

Based on the axioms A1 through A10 and B1 through B6, we need to guarantee that the receiver is able to read the message, but the bystander cannot. We also need to verify that the above axioms are consistent. These requirements are listed below.

**C1**  $knows(R, message)$ .

**C2**  $\neg knows(B, message)$ .

**C3** The formulae in A1-A10 and B1-B6 are consistent.

There are two advantages in the logic approach: First, we can check that the sender and the receiver can exchange messages securely. Second, in case the bystander manages to know the message, which means that something not discussed in the above analysis goes wrong, we could discover the potential leakages by studying all the formulae (*i.e.*, assumptions) that are used in the proof of the C2 formula above.

## 4 A defective protocol

In this section, we show that defects in a defected protocol can be discovered with our approach. The protocol studied in this section is dynamic and hence cannot be verified. We use the testing approach to discover a defect in the protocol.

This protocol is concerned with group communication. We will use a simplified example to explain the protocol. It should be straightforward to extend this example to more realistic cases. A communication community consists of four clients ( $C_1, C_2, C_3$ , and  $C_4$ ) and a server. The server is responsible for broadcasting encrypted messages to the four clients. The four clients are partitioned into two groups:  $C_1$  and  $C_2$  are in the first group and  $C_3$  and  $C_4$  are in the second group. As time goes by, current clients may leave the groups and new clients may join the group. In general, there may be more than two groups and a group may include other subgroups.

There are two kinds of messages broadcast over the open channel: the regular messages and the rekey messages. The regular messages carry out regular communication and the rekey messages are used for updating and distributing the various keys.

There is an overall key  $K_0$ , which is known to all clients as well as the server. There is a group key for each group. We use  $K_1$  and  $K_2$  to represent the two group keys. Each client also possesses a private key. We will use  $K_3, K_4, K_5$ , and  $K_6$  to represent the private keys of the four clients, respectively. The server knows all the keys. The following table shows the keys each client knows.

client	keys
$C_1$	$K_0, K_1, K_3$
$C_2$	$K_0, K_1, K_4$
$C_3$	$K_0, K_2, K_5$
$C_4$	$K_0, K_2, K_6$

In normal communication, the server encrypts the regular messages with the overall key  $K_0$  and broadcasts the encrypted messages to all the clients (as well as to all bystanders). Since only the clients are assumed to have the key  $K_0$ , only the clients can decrypt the messages.

New clients may join the group. Similarly, current clients may leave the group. This protocol is concerned with updating and distributing the keys when clients join or leave the groups. We will use examples to explain the process of joining and leaving a group.

When a client leaves the group, the server needs to change all the keys the leaving client knows and broadcasts the new keys (of course, in the encrypted form) to other clients. The change is necessary in order to prevent the leaving client from eavesdropping future communications. The messages carrying the new keys are called the *rekey messages*.

First suppose the client  $C_4$  leaves the group. The keys  $K_0, K_2$  and  $K_6$  become obsolete. The server generates new keys  $N_0$  and  $N_2$  to replace the obsolete keys  $K_0$  and  $K_2$ , respectively. The rekey message is made up of the new key  $N_0$  appended with the result of  $N_2$  encrypted with

$K_2$ . The rekey message is then encrypted with  $K_1$  and then is broadcast to clients in the first group (*i.e.*,  $C_1$  and  $C_2$ ). The rekey message is also encrypted with  $K_5$  and is then sent to clients in the second group (*i.e.*,  $C_3$ . Note that, in general, there might be other clients in the second group.) Vice versa for all other groups, if they exist.

**A1**  $public([N_0, [N_2]_{K_2}]_{K_1})$ .

**A2**  $public([N_0, [N_2]_{K_2}]_{K_5})$ .

Suppose that after  $C_4$  left the group, a new client  $D_4$  joins the group. The server places  $D_4$  in the same group as  $C_3$ . The server generates a new key  $N_6$  and passes  $N_6$  to client  $D_4$  through a separate, secure channel (say, over the telephone or by a registered postal mail). Later, the server encrypts the overall key  $N_0$  and the group key  $N_2$  with key  $N_6$  and sends the encrypted keys to  $D_4$  over the open channel.

**A3**  $public([N_0, N_2]_{N_6})$ .

Finally suppose that the client  $C_2$  leaves the group. The keys  $N_0$ ,  $K_1$ , and  $K_4$  become obsolete. The server generates new keys  $P_0$  and  $P_1$  to replace  $N_0$  and  $K_1$ , respectively. The server composes a rekey message by prepending the new key  $P_0$  to the result of encrypting the new key  $P_1$  with the old key  $K_1$ . The server encrypts the rekey message with  $N_2$  and broadcasts the final result, over the open channel, to the clients in the second group (*i.e.*,  $C_3$  and  $D_4$ ). The server also needs to encrypt the rekey message with key  $K_3$  and broadcasts the result to the clients in the first group (*i.e.*,  $C_1$ ). Vice versa for all other groups, if they exist.

**A4**  $public([P_0, [P_1]_{K_1}]_{N_2})$ .

**A5**  $public([P_0, [P_1]_{K_1}]_{K_3})$ .

Everything looks fine at this moment because it seems that the two clients who left the group—client  $C_2$  (having the keys  $K_0$ ,  $N_0$ ,  $K_1$ , and  $K_4$ ) and client  $C_4$  (having the keys  $K_0$ ,  $K_2$ , and  $K_6$ )—cannot not decrypt any future communications among the group members. However, this is just an illusion.

It is true that neither  $C_2$  nor  $C_4$  alone can decrypt future group communications. However, because the rekey message  $[N_0, [N_2]_{K_2}]_{K_1}$  is transmitted over the open channel, it is assumed that everybody, including  $C_2$  and  $C_4$ , can read and record (but not necessarily decrypt) the rekey message. Since  $C_2$  has the key  $K_1$  and  $C_4$  has the key  $K_2$ , the clients  $C_2$  and  $C_4$  are able to know the key  $N_2$  if they collaborate.

Furthermore, since the message  $[P_0, [P_1]_{K_1}]_{N_2}$  is also transmitted over the open channel, the clients  $C_2$  and  $C_4$  are able to grasp the key  $P_0$  with the key  $N_2$ . Once they have the key  $P_0$ , they are able to eavesdrop to all future group communications. Here we have discovered a defect in the security protocol.

The above protocol is a simplified version of a draft protocol studied in a previous research project. With additional mechanisms, the original protocol is able to create new groups and merge existing groups. For our discussion here, we can concentrate on the simplified protocol.

The logic approach is able to help the protocol designers to clarify potential defects in the protocol. For the above simplified protocol and the above scenario, we first decide that there are six participants (the server, the four clients, and the bystander) in the protocol. Since the server always knows all the keys, it is not necessary to include him in our logical framework. We use five constant symbols to represent the four clients ( $C_1$ ,  $C_2$ ,  $C_3$ , and  $C_4$ ) and the bystander ( $B$ ). The initial knowledge of the clients are represented by the following formulae:

**A6**  $knows(C_1, K_0)$ .

**A7**  $knows(C_1, K_1)$ .

**A8**  $knows(C_1, K_3)$ .

**A9**  $knows(C_2, K_0)$ .

**A10**  $knows(C_2, K_1)$ .

**A11**  $knows(C_2, K_4)$ .

**A12**  $knows(C_3, K_0)$ .

**A13**  $knows(C_3, K_2)$ .

**A14**  $knows(C_3, K_5)$ .

**A15**  $knows(C_4, K_0)$ .

**A16**  $knows(C_4, K_2)$ .

**A17**  $knows(C_4, K_6)$ .

When client  $C_4$  leaves the group, two new constant symbols ( $N_0$  and  $N_2$ ) are introduced that represent the new keys. The following two formulae will be added to the framework that represent the conditions that should be satisfied. If later it is found that the conditions are not satisfied, the protocol must be defective. The formulae A1 and A2, also added to the framework, state that the encrypted messages transmitted over the open channel are made public.

**A18**  $\neg knows(C_4, N_0)$ .

**A19**  $\neg knows(C_4, N_2)$ .

Furthermore, when a client leaves a group, it is safe (though pessimistic) to assume that his knowledge becomes public.

**A20**  $public(K_0)$ .

**A21**  $public(K_2)$ .

**A22**  $public(K_6)$ .

When a new client joins a group, a new symbol  $D_4$  representing the new client and a new symbol  $N_6$  representing the new private key for  $D_4$  are created. The following formulae are added to the framework:

**A23**  $knows(D_4, N_6)$ .

**A24**  $public([N_0, N_2]_{N_6})$ .

When client  $C_2$  leaves the group, two new constant symbols ( $P_0$  and  $P_1$ ) are introduced in the framework that represent the new overall key and the new group key. The following two formulae will be added to the framework that represent the conditions that should be satisfied. If later it is found that the conditions are not satisfied, the protocol must be defective.

**A25**  $\neg knows(C_2, P_0)$ .

**A26**  $\neg knows(C_2, P_1)$ .

Similarly, when a client leaves a group, it is safe (though pessimistic) to assume that his knowledge becomes public.

**A27**  $public(K_0)$ .

**A28**  $public(K_1)$ .

**A29**  $public(K_4)$ .

**A30**  $public(N_0)$ .

We also need additional formulae that defines the predicate *public* and the decryption capability. This protocol is assumed to make use of a symmetric encryption algorithm.

**A31**  $\forall(x)\forall(m)[public(m) \rightarrow knows(x, m)]$ .

**A32**  $\forall(x)\forall(m)\forall(k)[knows(x, \{m\}_k) \wedge knows(x, k) \rightarrow knows(x, m)]$ .

**A33**  $\forall(x)\forall(m)\forall(k)[knows(x, \{m\}_k) \wedge \neg knows(x, k) \rightarrow \neg knows(x, m)]$ .

**A34**  $\forall(x)\forall(m)\forall(k)[\neg knows(x, \{m\}_k) \rightarrow \neg knows(x, m)]$ .

Based on the above formulae A1 through A34, we can discover that the above formulae are inconsistent—we may derive both  $\neg knows(C_2, P_0)$  and  $knows(C_2, P_0)$ .

## 4.1 Summary of our approach

In this subsection, we briefly review our testing approach. To evaluate a protocol-analysis method, we first need to establish a security criteria of a protocol. For the protocol in Section 4, we adopt the following security criteria:

1. A bystander cannot read any message unless he is authorized.

2. A participant is able to read only the messages that he is entitled to.
3. The member who leaves the group is not able to know any future messages that he is not entitled to.
4. The member who joins the group is not able to know any previous messages that he is not entitled to.

These criteria are translated into logic formulae later.

Given a security protocol, we identify the roles involved in the protocol. There are two types of roles: the participants (such as senders, receivers, and servers) and the bystanders.

Then we list the initial key distribution—who knows what keys initially—as logic formulae. We assume that the initial keys are distributed over a secure channel.

For each network activity according to the protocol, new keys may be generated. They are represented as new constant symbols.

When a new participant joins the group, a new constant symbol is created to represent the new participant. When a participant retires from the communication group, we assume that his knowledge (primarily all the keys he knows) becomes public immediately.

When a message passes through the open channel, that message is also made public.

For the protocol in Section 4, new keys are generated and old keys become obsolete as time goes by. We generate a *trace* of the activities on the network that is relevant to the protocol under investigation. Sometimes, there are only a small number of traces; in other cases, there could be many or infinite traces. For each trace, we simulate the activities, generate new constant symbols and new formulae and check the consistency of the all the formulae at every stage.

A weakness of the above analysis method is that the analyzer needs to discover a scenario that shows the existence of a defect. However, with an appropriate protocol simulator, this weakness can be somehow alleviated.

Note that the constant symbols and formulae that are introduced into the logic framework when a client joins or leaves the community can be generated *automatically* with a protocol simulator. In addition, the traces (that is, the sequences of the leaving and joining activities) can also be generated automatically. What we need is a tool that can translate a protocol specification into a program that simulates the protocol activities.

## 5 Conclusion

In this paper, we use three examples to illustrate a new, general approach to the verification and testing of security protocols. Our approach is based on logic inference.

For the verification and testing of security protocols, we usually make the worst assumptions, such as

1. All the bystanders collaborate.

2. All the bystanders and all participants are constantly listening and recording all the communication traffic.
3. Anyone who left the group will immediately leak all the secrets he knew.

These worst assumptions are reasonable in that a security protocol is expected to withstand the fiercest attacks.

Because different security protocols are designed for different security objectives, it is impossible to set up generic security criteria for all security protocols. For example, the deniable ring authentication protocols [15], intentionally leak secrets. However, their objectives are to hide the message issuer or the message authenticator. We have to set up the security criteria for individual protocols separately.

From the point of view of protocol verification, there are two kinds of security protocols: static one and dynamic ones. The static protocols, such as those in the first and second examples, can be *verified* in a logically complete way. On the other hand, the dynamic protocols, such as the one in the third example, can only be *tested* by inventing problematic traces. For the dynamic protocols, exhausted testing is usually beyond the question. This situation is a little similar to software testing: Simple programs might be verified [16]; more complex programs can only be selectively tested [17].

**Acknowledgement.** The first author wishes to express his sincere gratitude to Dr. Lin Chyng-Yuh for her help during his research work. The work reported here is supported by National Science Council, Taiwan, R.O.C., under grant NSC 91-2213-E-009-068.

## References

- [1] 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.
- [2] J.G. Steiner, C. Neuman, and J.I. Schiller, "An authentication service for open network systems," in *Proceedings of the Winter 1988 USENIX Conference*, pp. 191-202, 1988.
- [3] A. Kehne, J. Schonwalder, and H. Langendorfer, "A nounce-based protocol for multiple authentication," *ACM Operating Systems Review*, Vol. 26, No. 4, pp. 84-89, October 1992.
- [4] C. Laferriere and R. Charland, "Authentication and authorization techniques in distributed systems," *Proceedings of IEEE 1993 International Carnahan Conference on Security Technology*, pp. 164-170, 1993.
- [5] B.C. Neuman and S.G. Stubblebine, "A note on the use of the timestamps as nounces," *ACM Operating Systems Review*, Vol. 27, No. 2, pp. 10-14, April 1993.
- [6] B. Clifford Neuman and Theodore Ts'o, "Kerberos: An authentication service for computer network," *IEEE Communications Magazine*, Vol. 32, No. 9, pp. 33-38, September 1994.
- [7] S.P. Shieh and W.H. Yang, "An authentication and key distribution system for open network systems," *ACM Operating Systems Review*, Vol. 30, No. 2, pp. 32-41, 1996.
- [8] 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.
- [9] W. Diffie and M.E. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, IT-22, pp. 644-654, November 1976.
- [10] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Trans. Computer Systems*, Vol. 8, No. 1, pp. 18-36, 1990.
- [11] R.E. Hodel, *An Introduction to Mathematical Logic*, PWS Publishing, Boston, 1995.
- [12] J. Kelly, *The Essence of Logic*, Prentice-Hall, New York, 1997.
- [13] A.A. Stolyar, *Introduction to Elementary Mathematical Logic*, Dover, New York, 1970.
- [14] C.A.R. Hoare, "An axiomatic basic for computer programming," *Communications of ACM*, Vol. 12, No. 10, pp. 576-580, October 1969.
- [15] M. Naor, "Deniable ring authentication," *Crypto 2002*, August 2002.
- [16] D. Gries, *The Science of Programming*, Springer-Verlag, New York, 1981.
- [17] R.S. Pressman, *Software Engineering*, 3rd ed., McGraw-Hill, New York, 1992.