# 行政院國家科學委員會專題研究計畫　期中進度報告

## 計算機網路之安全通信協定的驗證與系統實作(1/2)

計畫類別：個別型計畫
計畫編號：NSC93-2213-E-009-092-
執行期間：93 年 08 月 01 日至 94 年 07 月 31 日
執行單位：國立交通大學資訊科學學系(所)

計畫主持人：楊武

計畫參與人員：楊武

報告類型：精簡報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 94 年 5 月 16 日

# 行政院國家科學委員會專題研究計畫成果報告

計畫編號：NSC 93-2213-E-009-092-

執行期限：93年8月1日至95年7月31日

主持人：楊武　國立交通大學資訊科學學系

## Abstract

*Security protocols are indispensable in secure communication. We give an operational semantics of security protocols in terms of a Prolog-like language. With this semantics, we can uncover attacks on a security protocol that are possible with no more than a given number of rounds. Though our approach is exhaustive testing, the majority of fruitless search is cut off by selecting a small number of representative values that could be sent by an attacker. Hence, the number of scenarios is relatively small and our method is quite practical. Furthermore, our method not only reports possible attacks but also describes the attacks in great detail. This description would be very helpful to protocol designers and analyzers.*

*Keywords and phrases: computer communication, computer security, logic, Prolog, security protocol, verification.*

## 1 Introduction

To securely communicate with other parties, we need two things: First, we need an encryption algorithm so that nobody can decrypt an encrypted item unless he has the proper keys. Second, the *security protocol*[3] used for establishing the secret keys must be correct.

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. Given the rapid growth of distributed systems, there is a pressing need for a framework and tools for the development and analysis of security protocols [3].

Throughout this paper, we will use the following Neumann-Stubblebine protocol [7] to illustrate our method for testing security protocols. (The notation $\{X\}K$ denotes the result of encrypting $X$ with key $K$.)

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

$2.B \rightarrow S : B, \{A, Na, Tb\}Kbs, Nb.$

$3.S \rightarrow A : \{B, Na, Kab, Tb\}Kas, \{A, Kab, Tb\}Kbs, Nb.$

$4.A \rightarrow B : \{A, Kab, Tb\}Kbs, \{Nb\}Kab$

$A$ and $B$ are two roles who want to establish a shared secret key $Kab$ though a trusted server $S$. An attack on this protocol goes as follows: The attacker, say $diana$, pretends to be $alice$ and takes up the role $A$. She first sends $alice$'s identity (denoted by $A$) and a nonce $Na$ to $barbara$ ($barbara$ will play the role $B$). Note that $Na$ is generated by $diana$. When $barbara$ sends the second message to the server $charm$ (who will paly the role $S$) for verification, $diana$ will record the second item $\{A, Na, Tb\}Kbs$. Then $charm$ will send the third message to $alice$. $diana$ intercepts this message. However, $diana$ could not decrypt the items in the third message because she does not possess the keys $Kas$ and $Kbs$. $diana$ will send the two items $\{A, Na, Tb\}Kbs$ and $\{Nb\}Na$ to $barbara$ as the fourth message. At this stage, $barbara$ is fooled to believe $Na$ is the session key between her and $alice$. From now on, $diana$ will use $Na$ to ask $barabara$ for sensitive information.

A *round* of the protocol consists of exchanging the above four messages. Some attacks can be done in a single round (such as the one described above) and others may need more rounds. For example, an attack to Otway-Rees protocol, an attack to SSH [2] and the *parallel-session* attacks need more than one round. However, the number of rounds needed in an attack is quite small in practice. An attacker may impersonate a role in one round and another in a second round in order to collect necessary information.

We give an operational semantics of security protocols in terms of a Prolog-like language. With this semantics, attacks on a security protocol are uncovered by exhaustive testing. The exhaustive testing is *limited* in that (1) we are confined within a fixed number of rounds in a test session; (2) the attacker can only fake no more than a fixed number of bogus atomic values; and (3) the faked values that are sent by an attacker, such as $\{A, Kab, Tb\}Kbs$, must have the correct format, though the contents may not be genuine. In this way, the majority of the search space, of which most are very possibly fruitless, is cut off.

With the method discussed in this paper, it is easy to experiment with different numbers of rounds and bogus items and with different kinds of collaborations, such as a real principal collaborating with the attacker and 3-person collaboration. Our method is capable of discovering type-flaw attacks [5, 9]. Our method may be extended to investigate the interaction of two or more different security protocols.

In our method, we frequently need to compare two values for equality. To detect freshness attacks, it is necessary to determine if one nonce is greater than another. Our method can be easily extended to include this ability and hence is able to detect freshness attacks. Our method is applicable to both symmetric and asymmetric encryption.

Some security protocols make use of broadcast messages (for group communication), rather than one-to-one messages. Our current method is unable to handle broadcast messages because there are many recipients. In our method, we also assume that the attacker cannot intercept and change a message. The attacker(s) can only eavesdrop the messages on the communication channel and pretend to be an honest principal.

Security protocols are first studied in [6], which leads to the Kerberos system. Paulson [8] shows a logic method for proving the correctness of a security protocol. Yang and Tsay [10] discusses a simple method for proving the existence of certain flaws in security protocols. Their method is based on logic. Heather et al. [5] and Li et al. [9] report a tagging scheme for preventing type-flaw attacks.

Clark and Jacob also view a security protocol as a program [4] and the associated BAN logic as its semantics. Their work generates all the reasonable protocols, *executes* these protocols, and selects the good ones. Ours differs from theirs in that we use extended Prolog as the semantic language. Our approach generates and executes all reasonable attacks and checks if the protocol can be broken down.

## 2 Basic Constants and Predicates

Consider the Neuman-Stubblebine protocol. There are three roles: $A$, $B$, and $S$. In this section, we will consider five principals $alice$, $barbara$, $charm$, $dian$, and $elisa$. $diana$ and $elisa$ are collaborating attackers who attempt to cheat on $barbara$. (We use different attackers in different rounds of the protocol and allow them to collaborate.) Each honest principal has an identity, which is public. An attacker pretends to be an honest principal by using that principal's identity. (In what follows, $cut$ are used to cut off unnecessary search.)
$Principal(alice).\ Identity(alice, aliceID).$
$Principal(barbara).\ Identity(barbara, barbaraID).$
$Principal(charm).\ Identity(charm, charmID).$
$Principal(diana).\ Dishonest(dianan).$
$Principal(elisa).\ Dishonest(elisa).$
$Public(XID) : -Identity(X, XID).$
$Identity(X, YID) : -Pretend(X, Y), Identity(Y, YID).$
$Identity(\_, \_) : -!, fail.$

A person may play different roles in different rounds.
$Pretend(diana, alice).\ Pretend(elisa, alice).$
$Roles(diana, barbara, charm, 1).$
$Roles(barbara, elisa, charm, 2).$

Each list of $Roles(\_, \_, \_, \_)$ constitutes a *scenario*. We can exhaustively generate and examine every possible scenario (of course, under the restriction that at most a certain number of rounds will be considered). Fortunately, the number of reasonable scenarios are not too many.

We use symbolic constants to represent (existing and newly generated) keys, nonces, timestamps, *tokens*, and bogus items. We have facts and rules that describe the initial key distribution and the generation of nonces, timestamps, tokens and bogus items. *Tokens* are used to initiate messages. Suppose we will test two rounds and there are four messages per round. Thus, there are exactly 8 tokens, one for each message instance. (If we want to examine three rounds, there will be 12 tokens.) Increasing the number of rounds may discover more weaknesses while at the cost of more analysis time. Whenever a principal receives a token, he has the right (and obligation) to send out the corresponding message. Our convention is that the sender of the first message is the one who initiates a round of a protocol, who is the first parameter in the $Roles()$ predicate.

During an attack, the attacker may need to manufacture several bogus items. The number of bogus items needed per round is no more than the number of atomic values in all message, which is usually a small number. Our method allows a user to specify the number of bogus items he needs in testing a protocol. The more bogus items, the more possible an attack can be uncovered, however, at the cost of more analysis time.

$diana$ and $elisa$ may actually be the same attacker. It is more flexible to use different attackers in different rounds and to test different collaboration scenarios. Collaborators share their knowledge.
$Collaborate(diana, elisa).$
$Collaborate(elisa, diana).$
$Knows(A, X) : -Collaborate(A, B), Knows(B, X).$

With $Collaborate$, it is easy to examine what may happen if two or more principals and/or the attackers collaborate. Note that $Collaborate$ is not necessarily a symmetric relation.

In testing a protocol, we limit the scope of searching to *atomic* values and proper structures built with atomic values using the binary $enc$ (encryption) and $comp$ (composition) operations. Only the constants declared above are atomic. There is not any other constant symbol.

## 3 A Semantics of Security Protocols

A security protocol is similar to a traditional program in that both contain a list of instructions for execution. A traditional program emphasizes that the desirable result will be produced. On the other hand, it is usually quite obvious that a security protocol will achieve its goal of exchanging keys. The emphasis of a security protocol is that the attacker will not be able to know the key under given assumptions. Thus, in designing an operational semantics of security protocols, we have to model what the attacker is capable of knowing and doing.

A round of a security protocol usually consists of several messages. One message triggers another. The first message in a protocol is triggered by the initiator's intention to communicate with another principal. The following clauses use *tokens* to trigger another message. By the way, once a message is sent out on the network, we assume that all items in the message become public since there is generally no way to prevent an attacker from eavesdropping on the physical communication media.

For each message, the sender needs to receive the token for the message. Then he needs to demonstrate his ability to create all the items in the message. Finally, all items needs to pass the receiver's verification (of course, to the extent of the receiver's knowledge). When all is done, the token for the next message is issued. Here we will show the rules for the first message. Rules for other messages are similar.

1. $A \rightarrow B : A, Na.$
$\mathit{Completes}1(J, P11, P12) : -\mathit{Received}(X, T, J),$
$\mathit{Token}(T, 1, J)$, $\mathit{Roles}(X, \_, \_, J), \mathit{Item}(P11, 1, 1, J),$
$\mathit{Item}(P12, 1, 2, J).$
$\mathit{Received}(Y, S, J) : -\mathit{Completes}1(J, \_, \_),$
$\mathit{Roles}(\_, Y, \_, J), \mathit{Token}(S, 2, J).$
$\mathit{Public}(P11) : -\mathit{Completes}1(\_, P11, \_).$
$\mathit{Public}(P12) : -\mathit{Completes}1(\_, \_, P12).$

Next we will show the the clauses for the messages in the security protocol. We consider each item in each message in turn. There are four clauses for each item in a message, one for each of the following four cases: (1) both the sender and the receiver are honest; (2) the sender is dishonest but the receiver is honest; (3) the sender is honest but the receiver is dishonest; and (4) both the sender and the receiver are dishonest. A dishonest principal can also use the clause for the honest one if he decides not to cheat in a particular item.

In each clause, the left-hand side contains certain predicates. These predicates are classified into two groups: (1) the binding predicates, which the sender uses to create the item; and (2) the verification predicates (those in boxes), which the receiver uses to verify the components in the item. Note that not every component can be verified immediately by the receiver. For instance, an item encrypted with a key not known to the receiver, such as the second item, $\{A, Kab, Tb\}Kbs$, in the third message, which is encrypted with key $Kbs$, which is not known to the receiver $A$, cannot be verified by the receiver immediately. However, this encrypted item will eventually be decrypted and verified in a well-formed protocol.

A message consists of one or more items. An item is either an *atomic* value or a *composite* one. A composite value is made up of atomic values with the *enc* (encryption) and *comp* (composition) operations. We assume that an atomic item always contains an atomic value and a composite item always contains a composite value with the correct structure. When an attacker fakes an item, only the atomic values in the fake item are replaced with incorrect atomic values.

From the sender's side, the sender must be able to build an item from the values he knows with the necessary $enc$ and $comp$ operations. The values come from three sources:

- The value could be a priori knowledge, such as initial key distribution and principals' identities.
- The value could also be atomic values generated by the sender, such as a nonce, a timestamp, and a new key.
- The value could also be obtained from a previous message that the sender receives, such as $\{A, Kab, Tb\}Kbs$ in the fourth message in the Neuman-Stubblebine protocol. The sender simply passes this item to another principal.

From the receiver's side, the receiver must either verify the values in the item or pass the item to another principal. To verify a value, there are three ways:

- Verify that the value is a priori information, such as initial key distribution and principals' identities.
- Verify that the value was generated by the receiver previously, such as a nonce and a timestamp.
- Verify that the value is identical to a previously received value.

The attacker is entitled to fake the items in all messages sent by the impersonated principal in a single round. Anything that the attacker knows can fill in the empty slots in the item. Note that public knowledge can still be faked if it is verified after the goal of attack is achieved.

In what follows, there are four clauses for each item in each message. The right-hand side of each clause consists of two groups of predicates: binding and verification predicates. The verification predicates are enclosed in boxes. The text within the square brackets explains the individual predicates. Certain clauses can be further simplified easily.

Note that we may consider each item in a message separately because knowing $comp(X, Y)$ is exactly the same as knowing $X$ and knowing $Y$. This argument is not valid for encryption: in general, knowing $enc(comp(X, Y), K)$ is quite different from knowing $enc(X, K)$ and knowing $enc(Y, K)$ [1]. Here we list the clauses for the first message. Other messages are processed similarly.

1. $A \rightarrow B : A, Na.$
$\mathit{Item}(XId, 1, 1, J) : -\mathit{Roles}(X, \_, \_, J),$
$\mathit{Identity}(X, XId)$, [$A$ binds $XId$ to a constant symbol.]
$\boxed{\mathit{Roles}(X, \_, \_, J), \mathit{Identity}(X, XId).}$ [$B$ verifies $XId$.]
$\mathit{Item}(XId, 1, 1, J) : -\mathit{Dishonest}(X), \mathit{Roles}(X, \_, \_, J),$
$\mathit{Atomic}(XId), \mathit{Knows}(X, XId)$, [$A$ fakes $XId$.]
$\boxed{\mathit{Roles}(X, \_, \_, J), \mathit{Identity}(X, XId).}$ [$B$ verifies $XId$.]

$Item(XId, 1, 1, J) : -Roles(X, \_, \_, J)$,
$Identity(X, XId)$, [$A$ binds $XId$ to a constant symbol.]
$\boxed{Dishonest(Y), Roles(\_, Y, \_, J).}$ [$B$ is dishonest.]
$Item(XId, 1, 1, J) : -Dishonest(X)$,
$Roles(X, \_, \_, J), Atomic(XId), Knows(X, XId)$, [$A$ fakes $XId$.]
$\boxed{Dishonest(Y), Roles(\_, Y, \_, J).}$ [$B$ is dishonest.]
$Item(Na, 1, 2, J) : -$
$NonceNa(Na, J)$, [$A$ binds $Na$ to a constant symbol.]
$\boxed{.}$ [$B$ believes in $Na$. No verification.]
$Item(Na, 1, 2, J) : -Dishonest(X)$,
$Roles(X, \_, \_, J), Atomic(Na), Knows(X, Na)$, [$A$ fakes $Na$.]
$\boxed{.}$ [$B$ believes in $Na$. No verification.]
$Item(Na, 1, 2, J) : -$
$NonceNa(Na, J)$, [$A$ binds $Na$ to a constant symbol.]
$\boxed{\text{Dishonest(Y), Roles(\_, Y, \_, J).}}$ [$B$ is dishonest.]
$Item(Na, 1, 2, J) : -Dishonest(X)$,
$Roles(X, \_, \_, J), Atomic(Na), Knows(X, Na)$, [$A$ fakes $Na$.]
$\boxed{\text{Dishonest(Y), Roles(\_, Y, \_, J).}}$ [$B$ is dishonest.]

In order to fool a principal, the attacker only needs to fake all the messages that were sent to that principal in a single round. For example, to fool $B$, the attacker needs to successfully fake the first and the fourth messages in a single round. In addition, only the values that can be verified by $B$ need to be genuine. For the first message, only $XId$ needs to be genuine. For the fourth message, $Kbs$, $XId$, $Tb$ and $Nb$ needs to be genuine.

Most security protocols are used to establish a key shared between two or more principals. For the Neuman-Stubblebine protocol, the attacker succeeds when $B$ believes in a compromised key $Kab$. Thus, to attack $B$ in the first round, the attacker's goal is to steal $Kab$ which $B$ will always accept.

$$
\begin{aligned}
?- \; & Completes1(1, \_, \_), Completes4(1, \_, \_), \\
& Item(enc(comp(comp(\_, Kab), \_), \_), 4, 1, 1), \\
& Knows(diana, Kab).
\end{aligned}
$$

To allow the attacker to inject fake messages, we create the necessary tokens for him, by adding the $Received$ clauses which $diana$ may elect to use in her attack: $Received(diana, t11, 1), Received(diana, t14, 1)$.

## 4 Conclusion

We have given an operational semantics for a security protocol, which is ready for simulating the execution of the protocol. With extensive simulation, it is possible to uncover hidden flaws in a security protocol before it is put into practical use. Our method is very practical because it is able to cut off many fruitless search.

## References

[1] M. Abadi, R. Needham, "Prudent engineering practice for cryptographic protocols," *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, pp. 6-15, January 1996.

[2] M. Abadi, "Explicit communication revisited: two new attacks on authentication protocols," *IEEE Transactions on Software Engineering*, Vol. 23, No. 1, pp. 185 - 186, March 1997.

[3] J.A. Clark and J.L. Jacob, "A survey of authentication protocol literature: Version 1.0," November 1997, available at http://www-users.cs.york.ac.uk/ jac/.

[4] J.A. Clark and J.L. Jacob, "Protocols are Programs Too: the Meta-heuristic Search for Security Protocols," *Information and Software Technology* Vol. 43, 891-904, 2001.

[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] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Comm. ACM*, 21(12), 993-999, 1978.

[7] B.C. Neumann and S.G. Stubblebine, "A note on the use of timestamps as nonces," *ACM Operating Systems Reviews*, 27(2), 10-14, April 1993.

[8] L.C. Paulson, "Proving security protocols correct," In Proceedings of 14th Symposium on Logic in Computer Science, 370-381, 1999.

[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] W. Yang and C.-W. Tsay (2002), "A logic approach to the verification and testing of security protocols," In Proceedings of International Conference on Communcations and Computer Networks (CCN 2002) (Cambridge, MA, November 4-6), 140-145, 2002.