Verifying Cryptographic Protocols

Xiaoqi Ma[1] and Xiaochun Cheng[2]
Department of Computer Science, The University of Reading
Reading RG6 6AY, England, UK
[1]xiaoqi.ma@rdg.ac.uk
[2]x.cheng@rdg.ac.uk

Abstract - *Information security is a key issue of network communication. Existing cryptographic protocols usually contain flaws. This paper attempts to provide a new framework to verify these protocols. A many-valued logic is used to describe the knowledge and belief states of participating principles. A number of predicates and action functions are also used to model the network communication environment. Domain rules are given to describe the transitions of principals' knowledge and belief states. An example of public key authentication protocols has been studied to show the validity of the framework.*

Keywords: Many-valued logic, protocol verification, cryptographic protocol.

1 Introduction

Information security is important for communication over the Internet. To keep secrecy of sensitive information, cryptography and security protocols are extensively used. However, cryptographic protocols often have flaws. As a result, protocol verification becomes a research highlight.

There are two main categories of formal methods for protocol verification. One is the state-based methods, and the other is the rule-based methods. The first kind can verify most protocols, but the search spaces will become huge when the protocols are complex [8]. The methods of the second kind could be efficient, but they also have some problems [5]. For example, the BAN logic and the ENDL framework can only detect flaws in limited types of protocols and ignore others [2][8].

In this paper, we propose a new formal framework of cryptographic protocol verification. Our framework is based on a many-valued logic [3][4]. The set of truth values has been extended beyond the traditional {*true*, *false*} set. We use a four-valued logic with truth value set {*true*, *false*, *inconsistent*, *ignorant*} in this paper. Rules in the framework describe the conditions under which knowledge and belief states will be changed.

In this paper, we first give a brief overview of preliminary knowledge about lattice-valued logic, and then give a detailed description of our framework. Thereafter, we take the famous Needham-Schroeder public key authentication protocol [7][8] as an example. Then we discuss the non-monotonic property of the framework. The last section concludes the paper.

2 Preliminaries

Belnap proposed a four-valued logic [1] to deal with inference concerning inconsistent or incomplete information. His ideas have been extensively studied and then extended by Ginsburg to a *bilattice* structure [3][4]. In Ginsburg's work, a bilattice is defined as "a set equipped with two partial orders and a negation operation that inverts one of them while leaving the other unchanged; it has been suggested that the truth values used by inference systems should be chosen from such a structure instead of the two-point set {*t*, *f*}" [4]. A bilattice consists of two tightly combined lattices sharing same truth values whose number is usually larger than 2.

In cryptographic protocol verification, it is natural to use bilattice to represent the concept and degree of truth, belief and knowledge. The bilattice we use in this paper is the so-called "smallest nontrivial bilattice" and consists of four logical values: *true*, *false*, *inconsistent* and *ignorant* [4]. There are two different partial orders (and therefore two different "single" lattices) in this bilattice [9]:

- A *truth* lattice with the logical values true and false at the top and bottom, and with inconsistent and ignorant between them, respectively.

- A *knowledge* lattice with the logical values inconsistent and ignorant at the top and bottom, and with true and false between them, respectively.

This structure can be depicted as the Hasse diagram [4][9] in Figure 1.

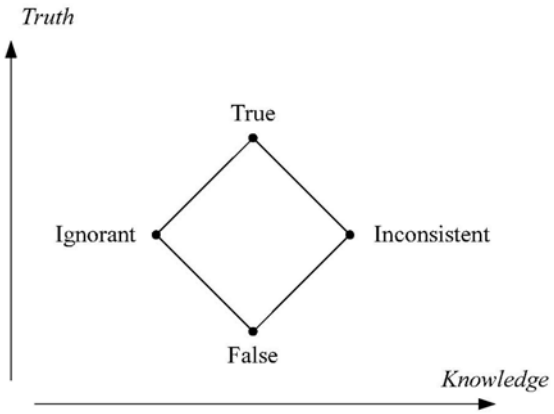

Figure 1. Hasse diagram of the bilattice

This paper uses the truth part of the bilattice. The truth lattice has the same connective set as traditional logic, although the semantics and truth tables are quite different. The truth tables for the three basic connectives $\wedge$ (and), $\vee$ (or) and $\neg$ (not) are given in Table 1 [9]. To save space, we use *T*, *F*, *U* and *C* instead of True, False, Ignorant and Inconsistent, respectively.

Table 1. Truth tables of *and*, *or* and *not* for the lattice

$\wedge$	T	F	U	C
T	T	F	U	C
F	F	F	F	F
U	U	F	U	F
C	C	F	F	C

(a) Truth table for the connective *and*

$\vee$	T	F	U	C
T	T	T	T	T
F	T	F	U	C
U	T	U	U	T
C	T	C	T	C

(b) Truth table for the connective *or*

$\neg$	
T	F
F	T
U	U
C	C

(c) Truth table for the connective *not*

The commutative, associative and distributive laws hold for $\wedge$ and $\vee$ as well as the *De Morgan* laws and the double negation law [9].

There are universal and existential quantifiers in the logic. The universal quantifier $\forall$ ("for all") and the existential $\exists$ ("there exists") has similar syntax and semantics as those in first-order predicate logic.

3 The verification framework

Our framework is composed of a number of basic notations, predicates, action functions, assumptions and rules, which are described as follows.

3.1 Basic notations

Individuals taking part in network communications are called *principals*, denoted with uppercase letters. Conventionally, *A*, *B*, …, stand for "honest" principals, while *I* stands for the *intruder*, or, in some literatures, the *adversary* or the *spy*, which are synonyms in our context.

Random numbers chosen by principals serve as *nonces* to identify protocol runs uniquely and avoid replay attack [8]. Nonces are normally denoted as N_a, N_b, etc, where the subscripts imply the producers of the nonces.

Every principal has some keys. In public key cryptosystem, the principal *A* basically has a *public key* and a corresponding *private key*, which are denoted as K_a and K_a^{-1}, respectively. On the other hand, in symmetric key cryptosystem, two communicating principals *A* and *B* normally share a session key, denoted as K_{ab}.

A *message* is a piece of information sent from one principal to another. A message can consist of names of principals, keys, nonces, or the combination of them. Compound messages are bracketed using curly braces, such as $\{A, N_a\}$. A compound message consisting more than two components can be understood as nested compound message. For example, $\{M_1, M_2, M_3\}$ is the abbreviation of $\{\{M_1, M_2\}, M_3\}$. A message can be encrypted or signed with a certain key. An encrypted message is attached with the key as the subscript. For example, a message *M* encrypted with *B*'s public key can be written as M_{K_b}. To gain clarity, we usually add curly brackets to the encrypted message, such as $\{M\}_{K_b}$.

3.2 Predicates

Several predicates are used in our framework to describe certain belief and knowledge states.

The predicate *Know*(*X*, *M*) describes *X*'s knowledge state about message *M*. If *X* knows that *M* is true, then the logical value of the predicate is true; if *X* knows that *M* is false, then its logical value is false; if *X* has no information about the truth of *M*, then its logical value is ignorant; if *X* has both information to conclude that *M* is true *and* false, then its logical value is inconsistent.

Similarly, the predicate *Auth*(*X*, *Y*, *M*) describes *X*'s authentication state about *Y* on message *M*, that is, whether message *M* is sent by *Y* to *X* and unmodified.

3.3 Action functions

A cryptographic protocol can be regarded as a series of message sending behaviours between different principals. We introduce action functions to describe this kind of behaviours. An action function *Send*(*X*, *Y*, *M*) means that the principal *X* sends the message *M* to another principal *Y*.

Accordingly, an action function *Rcv*(*X*, *M*) means that the principal *X* receives the message *M* from another.

3.4 Assumptions

Our inference system is based on some assumptions, which are widely accepted by most researchers in this field.

- In public cryptosystems, the public key of any principal is known to all other principals, while its private key is initially secret from others. Formally,

$$\forall X.\, \forall Y.\, (Know(X, K_Y) \equiv \mathrm{T})$$

$$\forall X.\, \forall Y.\, (\mathrm{X} \neq \mathrm{Y} \rightarrow (Know(X, K_Y^{-1}) \equiv \mathrm{U}))$$

- In symmetric cryptosystems, the session key shared between two principals is initially secret and unknown by all other principals (except the key server who knows all the keys in the whole system).

- The intruder always observes all messages sent through the network. He tries to use all the keys he knows to decrypt the messages on the network and send forged messages to others. He can also intercept messages sent from one principal to another. That is, the intruder has the "full" control over the network.

- There is only one intruder in the network. We always use *I* to denote the intruder.

- The intruder cannot read an encrypted message without the corresponding decryption key; i.e. secret keys are unguessable.

- An honest principal only read information addressed to him.

- A principal never sends messages to himself.

- Nonces are always different from each other.

3.5 Rules

A group of inference rules have been introduced into our framework to infer new knowledge from the old. We adopt some basic ideas from [2] but they are essentially different. All these rules can be divided into four categories:

(1) Encryption/Decryption rules

1.1 $Know(X, M) \wedge Know(X, K) \rightarrow Know(X, M_K)$

When a principal knows a message and a key, he can use this key to encrypt (sign) this message and get the encrypted (signed) message.

1.2 $Know(X, M_K) \wedge Know(X, K^{-1}) \rightarrow Know(X, M)$

When a principal knows a message encrypted (signed) with a key and the reverse of the key, he can use the reverse of the encryption (signature) key to get the original message. In public key cryptosystem, the public and private keys of a principal are reverses of each other. In symmetric cryptosystem, the reverse of a session key between two principals is itself (or a simple function of itself).

(2) Message combination/separation rules

2.1 $Know(X, M_1) \wedge Know(X, M_2) \rightarrow Know(X, \{M_1, M_2\})$

When a principal knows two messages, he can know the combination of them.

2.2 $Know(X, \{M_1, M_2\}) \rightarrow Know(X, M_1) \wedge Know(X, M_2)$

When a principal knows the combination of two messages, he can know them separately. These two rules can be used inductively to deal with compound messages consisting of more than two components.

(3) Message sending/receiving rules

3.1 $Send(X, Y, M) \rightarrow Rcv(Y, M)$

If a principal sends a message to another one, the object principal will eventually receive it.

3.2 $Send(X, Y, M) \rightarrow Rcv(I, M)$

As one of our assumptions describes, the intruder *I* can observe all information flowing over the network.

3.3 $Rcv(X, M) \rightarrow Know(X, M)$

After a principal receive a message, he will know it.

(4) Authentication rule

4.1 $Know(X, M) \wedge Know(Y, M) \wedge \forall Z.((Z \neq X \wedge Z \neq Y) \rightarrow (Know(Z, M) \equiv U) \wedge Rcv(X, M) \rightarrow Auth(X, Y, M)$

When only two principals *X* and *Y* know a certain message (i.e. no other principals know it), and *X* received it from other principal (remember our assumption that a principal never sends messages to himself), he can authenticate that the message was sent by *Y* and unmodified.

4.2 $\forall X, Y.(\exists Z.((Z \neq X \wedge Z \neq Y) \wedge Know(Z, M)) \rightarrow (Auth(X, Y, M) \equiv F))$

If a message has been divulged to a third party other than *X* and *Y*, then *X* will not authenticate that the message was sent by *Y* and unmodified.

3.6 Protocol modeling

A cryptographic protocol can be modeled as a series of *send* action functions, each meaning that a principal sends a message to another one.

4 Case study

An example is given here to describe how to use our framework to verify cryptographic protocols.

4.1 Needham-Schroeder public key authentication protocol

Originally, the Needham-Schroeder public key protocol [7] involves seven steps, four of which are concerning public key distribution procedures. In our model, all principals' public keys are open and known to the entire world. Therefore, the protocol can be simplified to only three steps [6] as follows.

1. $A \rightarrow B$: $\{N_a, A\}_{K_b}$
2. $B \rightarrow A$: $\{N_a, N_b\}_{K_a}$
3. $A \rightarrow B$: $\{N_b\}_{K_b}$

4.2 Lowe's fix to the Needham-Schroeder protocol

Lowe found a flaw [6] in the Needham-Schroeder public key protocol seventeen years after it was published. Lowe fixed the flaw by adding principal *B*'s name into the second message sent from *B* to *A* [6]. This fix has been proved to be correct and the new protocol is now secure [8]. The fixed protocol looks like as follows.

1. $A \rightarrow B$: $\{N_a, A\}_{K_b}$
2. $B \rightarrow A$: $\{N_a, N_b, B\}_{K_a}$
3. $A \rightarrow B$: $\{N_b\}_{K_b}$

4.3 Modeling the Needham-Schroeder protocol

The original protocol can be easily modeled as three message sending actions.

NS-1: $Send(A, B, \{N_a, A\}_{K_b})$

NS-2: $Send(B, A, \{N_a, N_b\}_{K_a})$

NS-3: $Send(A, B, \{N_b\}_{K_b})$

4.4 Verifying the Needham-Schroeder protocol

We formalise our assumptions, protocol steps, and assertions into logical formulae. At the very beginning, every principal knows all other's public key, but does not know any private key except that of himself.

(1)	$Know(A, K_b) \equiv T$	Assumption
(2)	$Know(B, K_a) \equiv T$	Assumption
(3)	$Know(I, K_a) \equiv T$	Assumption
(4)	$Know(I, K_b) \equiv T$	Assumption
(5)	$Know(A, K_i) \equiv T$	Assumption
(6)	$Know(A, K_a^{-1}) \equiv T$	Assumption
(7)	$Know(B, K_b^{-1}) \equiv T$	Assumption
(8)	$Know(I, K_i^{-1}) \equiv T$	Assumption
(9)	$Know(I, \{N_b\}_{K_b}) \equiv U$	Assumption

Then the protocol starts. Due to symmetry, we only need to consider the case that the principal *A* initialises the

protocol. In *A*'s view, he sends a message to another principal he wants to talk to. However, there are still two cases: (1) the message is sent to an honest principal, say, *B*, and (2) the message is sent to the intruder *I*.

In the first case, we can prove that the protocol can be carried on to the end. But this case does not explicitly concern the intruder, so we will not know whether the intruder can break the protocol. Therefore, we should consider the second case carefully. The following formulae describe the run of the protocol in this case.

(10)	$Send(A, I, \{N_a, A\}_{K_i}) \equiv \mathrm{T}$	NS-1 (run 1)
(11)	$Rcv(I, \{N_a, A\}_{K_i}) \equiv \mathrm{T}$	[3.1] (10)
(12)	$Know(I, \{N_a, A\}_{K_i}) \equiv \mathrm{T}$	[3.3] (11)
(13)	$Know(I, \{N_a, A\}) \equiv \mathrm{T}$	[1.2] (8)(12)
(14)	$Know(I, N_a) \equiv \mathrm{T}$	[2.2] (13)

By now, *I* has successfully got *A*'s nonce N_a. He can then use it to produce fake message and send it to another honest principal *B* to initialise a new session, impersonating *A*.

(15)	$Send(I, B, \{N_a, A\}_{K_b}) \equiv \mathrm{T}$	NS-1 (run 2)
(16)	$Rcv(B, \{N_a, A\}_{K_b}) \equiv \mathrm{T}$	[3.1] (15)
(17)	$Know(B, \{N_a, A\}_{K_b}) \equiv \mathrm{T}$	[3.3] (16)
(18)	$Know(B, \{N_a, A\}) \equiv \mathrm{T}$	[1.2] (7)(17)
(19)	$Know(B, N_a) \equiv \mathrm{T}$	[2.2] (18)
(20)	$Send(B, I, \{N_a, N_b\}_{K_a}) \equiv \mathrm{T}$	NS-2 (run 2)
(21)	$Rcv(I, \{N_a, N_b\}_{K_a}) \equiv \mathrm{T}$	[3.1] (20)
(22)	$Know(I, \{N_a, N_b\}_{K_a}) \equiv \mathrm{T}$	[3.3] (21)

Since *I* cannot decrypt the message $\{N_a, N_b\}_{K_a}$, his only choice is to send it to *A* and use *A* as an oracle.

(23)	$Send(I, A, \{N_a, N_b\}_{K_a}) \equiv \mathrm{T}$	NS-2 (run 1)
(24)	$Rcv(A, \{N_a, N_b\}_{K_a}) \equiv \mathrm{T}$	[3.1] (23)
(25)	$Know(A, \{N_a, N_b\}_{K_a}) \equiv \mathrm{T}$	[3.3] (24)
(26)	$Know(A, \{N_a, N_b\}) \equiv \mathrm{T}$	[1.2](6) (25)
(27)	$Know(A, N_b) \equiv \mathrm{T}$	[2.2] (26)
(28)	$Send(A, I, \{N_b\}_{K_i}) \equiv \mathrm{T}$	NS-3 (run 1)
(29)	$Rcv(I, \{N_b\}_{K_i}) \equiv \mathrm{T}$	[3.1] (28)
(30)	$Know(I, \{N_b\}_{K_i}) \equiv \mathrm{T}$	[3.3] (29)
(31)	$Know(I, N_b) \equiv \mathrm{T}$	[2.2] (8)(30)
(32)	$Know(I, \{N_b\}_{K_b}) \equiv \mathrm{T}$	[1.1] (4)(31)
(33)	$Auth(B, A, \{N_b\}_{K_b}) \equiv \mathrm{F}$	[4.2] (32)

Therefore, *B* will not authenticate that $\{N_b\}_{K_b}$ is sent by *A* and unmodified. It indicates a flaw in the Needham-Schroeder public key protocol. Actually, N_b is the nonce helping *B* identify *A* and should be kept secret from all other principals, especially the intruder. However, the inference step (31) shows that N_b has been divulged to *I*. This implies the flaw in the protocol.

On the contrary, in the fixed Needham-Schroeder public key protocol, this will not happen. The steps (1) to (19) are same to the above case. Steps from (20) are as follows.

(20')	$Send(B,I,\{N_a,N_b,\mathrm{B}\}_{K_a}) \equiv \mathrm{T}$	NS-2 (run 2)
(21')	$Rcv(I, \{N_a, N_b, \mathrm{B}\}_{K_a}) \equiv \mathrm{T}$	[3.1] (20')
(22')	$Know(I,\{N_a,N_b,\mathrm{B}\}_{K_a}) \equiv \mathrm{T}$	[3.3] (21')
(23')	$Send(I,A,\{N_a,N_b,\mathrm{B}\}_{K_a}) \equiv \mathrm{T}$	NS-2 (run 1)
(24')	$Rcv(A, \{N_a, N_b,\mathrm{B}\}_{K_a}) \equiv \mathrm{T}$	[3.1] (23')
(25')	$Know(A,\{N_a,N_b,\mathrm{B}\}_{K_a}) \equiv \mathrm{T}$	[3.3] (24')
(26')	$Know(A,\{N_a,N_b,\mathrm{B}\}) \equiv \mathrm{T}$	[1.2](6)(25')

When the verification proceeds to step (25), *A* will find the inconsistency between the name of *I* and the name in the received message, and he will then interrupt the session.

Therefore, N_b will not be sent out by *A* and divulged [8]. So the logical statement $Know(I, \{N_b\}_{K_b}) \equiv \mathrm{U}$ will remain true. Then the inference rule 4.1 can be safely used and the conclusion $Auth(B, A, \{N_b\}_{K_b})$ will be achieved, indicating the correctness of the fixed protocol.

5 Non-monotonic Property

Our framework is non-monotonic. The knowledge and belief of a principal keep changing during the process of inference.

At the beginning of a new run of a protocol, some assumptive statements and assertions are given. These statements are actually based on incomplete information, since at that time the protocol has not started yet and we do not know what will happen. Unlike monotonic inference, these statements and assertions do not necessarily always keep static in the inference. With the inference going on, some assertions may be changed.

As an example, in the verification of the Needham-Schroeder public key protocol, *I* does not know *B*'s nonce N_b at first, and we assume *Know*(*I*, $\{N_b\}_{K_b}$) ≡ U in the step (9) of the example. After some steps, *I* gets the secret number N_b by using *A* as an oracle to decrypt the encrypted message $\{N_a, N_b\}_{K_a}$. Therefore the assumption *Know*(*I*, $\{N_b\}_{K_b}$) ≡ U will not hold any longer. Instead, we will have *Know*(*I*, $\{N_b\}_{K_b}$) ≡ T in the step (32) of the example. This shows that the temporary results could be "updated".

6 Conclusions

In this paper, we have presented a new framework to verify cryptographic protocols. The framework is based on many-value logic, which contains four logic values: true, false, inconsistent and ignorant. We have introduced a number of rules to describe the knowledge and belief states of principals and the relationships among them. These rules give the conditions under which the knowledge and belief states can be changed, and how they can change. We have presented the verification process of the simplified Needham-Schroeder public key authentication protocol and of the fixed counterpart suggested by Lowe. These examples show how to use our framework.

Acknowledgements

Our researches had been supported by EC, EPSRC, the National Natural Science Foundation of China, and Hong Kong K C Wang Education Foundation.

References

[1] N. D. Belnap, Jr., "A useful four-valued logic". Dunn, J. M. and Epstein, G., editors, *Modern Uses of Multiple-Valued Logic*, pp. 8-37, 1977.

[2] Qingfeng Chen, "The verification logic for secure transaction protocols", PhD dissertation, University of Technology, Sydney, 2004.

[3] Matthew L. Ginsberg, "Multivalued logics: a uniform approach to reasoning in artificial intelligence", *Computational Intelligence*, Vol 4, pp. 265-316, 1988.

[4] Matthew L. Ginsberg, "Bilattices and Modal Operators", *Journal of Logic and Computation*, Vol 1, No. 1, July 1990.

[5] Armin Liebl. "Authentication in Distributed Systems: A Bibliography", *Operating Systems Review*, Vol 27, No. 4, pp. 122-136, October 1993.

[6] Gavin Lowe, "An attack on the Needham-Schroeder public-key authentication protocol", *Information Processing Letters*, Vol 56, No. 3, pp 131-133, 1995.

[7] Roger Needham and Michael Schroeder, "Using encryption for authentication in large networks of computers", *Communications of the ACM*, Vol 21, No. 12, pp. 993-999, 1978.

[8] Lawrence C. Paulson, "The inductive approach to verifying cryptographic protocols", *Journal of Computer Security*, Vol 6, No.1, pp. 85-128, September 1998.

[9] Jorgen Villadsen, "Paraconsistent knowledge bases and many-valued logic", Proceedings of the International Baltic Conference on Database and Information Systems, Tallinn, Estonia, pp. 77-90, 2002 (Revised).