Verifying security protocols by knowledge analysis

Xiaoqi Ma*
School of Systems Engineering,
The University of Reading, UK
E-mail: xiaoqi.ma@reading.ac.uk
*Corresponding author

Xiaochun Cheng
School of Systems Engineering,
The University of Reading, UK
E-mail: x.cheng@reading.ac.uk

Abstract: This paper describes a new interactive method to analyse knowledge of participants involved in security protocols and further to verify the correctness of the protocols. The method can detect attacks and flaws involving interleaving sessions besides normal attacks. The implementation of the method in a generic theorem proving environment, namely Isabelle, makes the verification of protocols mechanical and efficient; it can verify a medium-sized security protocol in less than ten seconds. As an example, the paper finds the flaw in the Needham-Schroeder public key authentication protocol and proves the secure properties and guarantees of the protocol with Lowe's fix to show the effectiveness of this method.

Keywords: security protocol; knowledge-based system; formal verification.

Reference to this paper should be made as follows: Ma, X. and Cheng, X. (2007) 'Verifying security protocols by knowledge analysis', Int. J. Security and Networks, Vol. x, No. x, pp.xx–xx.

Biographical notes: Xiaoqi Ma is a PhD student at The University of Reading. His research interests include computer security, formal methods, operating systems and real time systems.

1 INTRODUCTION

Information security is playing an increasingly important role in modern society, driven especially by the uptake of the Internet for information transfer. In certain areas, this issue is even vital. For example, military or security departments of governments often transmit a great amount of top-secret data, which, if divulged, could become a huge threat to the public and to national security. Even in our daily life, it is also necessary to protect important information. Consider e-commerce system as an example. No one is willing to purchase anything over the Internet before being assured that all their personal and financial information will always be kept secure and will never be leaked to any unauthorised person or organisation.

To achieve secrecy of sensitive information, cryptography is extensively used. However, cryptography can only offer protection if it is used in an appropriate way. Guidelines about how to use cryptographic technology are incorporated in the large number of security protocols that have been developed. These protocols in general contribute greatly to information security.

However, many such protocols are inherently incorrect, or at least cannot achieve completely all that is intended of them. Indeed, some protocols considered perfect at first were found to have flaws in them many years later. To evaluate and verify security protocols in a systematic way, Burrows *et al.* proposed their BAN logic (Burrows, 1990) in 1989. Subsequently, significant research work has been conducted in the area of designing formal methods for analysis of cryptographic protocols, and many good models have been proposed. Generally speaking, all these methods can be broadly divided into two categories, namely state based methods and rule based methods.

State based methods model security protocols as finite state machines. They search the state space exhaustively to see whether all the reachable states are safe (Paulson,

1997a). If some reachable state in a security protocol is proved to be unsafe, a flaw may be reported; otherwise, the protocol will be said to be correct and safe. Recognised state based methods include Lowe's CSP method with FDR as the model checker (Lowe, 1996) and Meadows' NRL Protocol Analyzer (Meadows, 1996a,b). State based methods are effective and many attacks have been found by this kind of methods. However, it is difficult to effectively control the size of the state space; when the protocol is large, the state space will become surprisingly huge and it will be extremely time consuming, or even practically impossible, to search the whole space.

Rule based methods formally express what principals can infer from messages received (Paulson, 1997a). With these approaches, the protocols, the necessary assumptions and the goals of the protocols are formulated in formal logic. Some specific properties of the protocols can be proved by using the axioms and rules of the logic (Liebl, 1993). Since rule based methods do not have to search large state space, they can normally converge very quickly, typically in less than one hundred steps for a medium sized protocol. In recent years, a number of rule based methods have been developed. Among them, the BAN logic introduced by Burrows *et al.* (Burrows, 1990), Bolignano's model (Bolignano, 1996) and Chen *et al.*'s ENDL framework (Chen, 2004, 2005) are good representatives. Paulson's inductive method (Paulson, 1997a,b, 1998) serves as a good example of using mathematical reasoning. Rule based methods are generally much more efficient than state exploration methods, and they do have found many subtle flaws. However, some of them only consider single runs of the protocols and usually ignore the interleaving of two or more sessions. So there are a lot of flaws which cannot be found with these logics.

To gain effectiveness from state based methods and efficiency from rule based methods, and overcome their drawbacks, we propose in this paper a new security protocol verification method, which is based on a knowledge-based framework. The method analyses the knowledge of participating principals and infers what they can know and can never know (Cheng, 2005; Ma, 2005a). It takes into consideration protocols concerning multiple interleaving sessions. By avoiding searching large state space and by implementing the method in a mechanical reasoning platform, Isabelle (Nipkow, 2003), it can be used to mechanically verify cryptographic protocols and can converge very quickly, providing high efficiency. The Needham-Schroeder public key authentication protocol (Needham, 1978) and Lowe's fix (Lowe, 1995) is considered in this paper as an example to show the effectiveness and efficiency of our method.

Section 2 gives a brief overview of the knowledge-based framework and its Isabelle implementation. The Needham-Schroeder public key authentication protocol and Lowe's fix are then introduced in section 3. We verify the original protocol mechanically and find the flaw in it in section 4, and then prove the correctness of the fixed protocol in section 5. Section 6 discusses relevant issues.

2 THE KNOWLEDGE BASED METHOD

This method focuses on the knowledge of all participants in the protocol. We describe their initial knowledge and infer what they can know and can never know with the progress of the protocol, processing this knowledge in formal logic. In other words, it concerns the knowledge analysis of all participants. This method can be implemented in Isabelle to enable mechanical verification.

2.1 Basic Notations and Data Structures

We call all participants taking part in network communications *principals*. They can be divided into three categories: the *server*, the *friends*, which stand for all "honest" principals, and the *spy*, which is also called the *adversary* or *intruder* in some literatures. In Isabelle notation, they can be described as follows:

```
datatype principal = Server | Friend nat | Spy
```

In the above definition, different friends are represented as different natural numbers.

Random numbers chosen by principals serve as *nonces* to identify protocol runs uniquely and avoid replay attack (Paulson, 1998). Every principal has some keys, such as *public encryption key* and *private signature key*. All the nonces and keys are simply represented as natural numbers.

One of the most important concepts is the *message*. A message is a piece of information sent from one principal to another. A message can consist of names of principals, nonces, keys, encrypted messages, signed messages, hashed messages, or a combination of these. It is recursively defined in Isabelle as follows:

```
datatype message = Principal principal
                 | Nonce nonce
                 | Key key
                 | MPair message message
                 | Encrypt message key
                 | Sign message key
                 | Hash message
```

Paired messages can be abbreviated using curly braces. For example, `MPair M1 M2` can be written as {`M1, M2`}. A compound message consisting of more than two components can be understood as a nested compound message. For example, {`M1, M2, M3`} is the abbreviation of {`M1,` {`M2, M3`}}.

2.2 Functions and Predicates

We define a number of useful *functions* and *predicates*. Among them are *key functions*. Function `Kpb` has the type `principal` ⇒ `key`, mapping a principal to its public encryption key. Functions `Kpv`, `Spb` and `Spv` similarly map a principal to its private encryption key, public signature key and private signature key, respectively. Both `Kpb` and `Spb` are injective functions.

Function *Nonce_of* maps a principal to its nonce. It is also an injective function.

To determine whether a message is a part of another one, we introduce a function *msg_part*. For a non-compound message, only itself is a part of it. For a compound message, message $M1$ is part of message $M2$, if and only if the component set of $M1$ is the subset of the component set of $M2$.

The predicate *Know* has the type *principal* $\Rightarrow$ *message* $\Rightarrow$ *bool*, describing a principal's knowledge state about a certain message. Similarly, the predicate *Auth* with the type *principal* $\Rightarrow$ *principal* $\Rightarrow$ *message* $\Rightarrow$ *bool* describes the first principal's authentication state about the second one on a certain message, that is, whether the message is sent by the second principal to the first one and is unmodified.

To describe cryptographic protocols, two *action functions* need to be introduced. One is *Send* with the type *principal* $\Rightarrow$ *principal* $\Rightarrow$ *message* $\Rightarrow$ *bool*, representing that one principal sends a message to another principal. Correspondingly, the other function is *Rcv* with the type *principal* $\Rightarrow$ *message* $\Rightarrow$ *bool*, meaning that a principal receives a certain message from others.

2.3 Assumptions

Our method is based on a number of assumptions, which are widely accepted by most researchers in this field.

The most important assumptions are *key assumptions*. In public cryptosystems, the public key of any principal is known to all other principals:

axioms
```
KeyAssump1 : "∀X Y. Know X (Key (Kpb Y))"
```

On the contrary, any principal's private key is initially secret from others except itself:

```
KeyAssump2 : "∀X. Know X (Key (Kpv X))"

KeyAssump3 :
   "∀X Y. ((X≠Y) ⟶ ¬(Know X (Key (Kpv Y))))"
```

We also assume that any principal's name is open to the world:

axioms
```
KnowName : "∀X Y. Know X (Principal Y)"
```

Besides that, any principal knows its own nonce:

axioms
```
KnowNonce : "∀X. Know X (Nonce (Nonce_of X))"
```

According to the definition of spy, it is not an honest principal, so we have:

axioms
```
Honest_not_Spy : "∀n. Spy ≠ Friend n"
```

In addition, we also have some other assumptions, which are described as follows:

- The spy always observes all messages sent through the network. It tries to use all the keys it knows to decrypt the message on the network and to send forged messages to others. It can also get messages sent from one principal to another. That is, the spy has the "full" control over the network.
- There is only one spy in the network. A single spy described above can be as powerful as multiple spies (Paulson, 1997a).
- The spy cannot read an encrypted message without the corresponding decryption key; i.e. secret keys are not guessable.
- An honest principal only reads information addressed to it. It never reads messages of which it is not the intended recipient.
- A principal never sends messages to itself. Sending a message to itself is meaningless.
- Nonces are always different from each other. They are random numbers used to prevent replay attacks.

2.4 Rules

We introduce a group of inference rules into the method to infer new knowledge from the old. All these rules can be divided into four categories.

The first is the *encryption/decryption rule*, which includes two rules.

When a principal knows a message and a key, it can use this key to encrypt the message and get the encrypted message:

```
Rule1_1 : "Know X M ∧ Know X (Key K)
                ⟹ Know X (Encrypt M K)"
```

When a principal knows a message encrypted with a key and the reverse of the key, it can use the reverse of the key to get the original message:

```
Rule1_2 : "Know X (Encrypt M K)
          ∧ Know X (Key (invKey K)) ⟹ Know X M"
```

The second category is the *message combination/separation rule*, which also includes two rules.

When a principal knows two messages, it can know the combination of them:

```
Rule2_1 : "Know X M1 ∧ Know X M2
                ⟹ Know X {M1, M2}"
```

When a principal knows the combination of two messages, it can know them separately:

```
Rule2_2 : "Know X {M1, M2}
```

```
⟹ Know X M1 ∧ Know X M2"
```

These two rules can be used inductively to deal with compound messages consisting of more than two components.

The third category is the *message sending/receiving rule*, which includes four rules.

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

```
Rule3_1 : "Send X Y M ⟹ Rcv Y M"
```

As one of our assumptions describes, the spy can observe all information flowing over the network:

```
Rule3_2 : "Send X Y M ⟹ Rcv Spy M"
```

After a principal receives a message, it will know it:

```
Rule3_3 : "Rcv X M ⟹ Know X M"
```

If a principal sends a message to another one, it must know it first:

```
Rule3_4 : "Send X Y M ⟹ Know X M"
```

The fourth category is the *authentication rule*, which still includes two rules.

If principal Y knows a message encrypted with principal X's public key, X receives this encrypted message, and all other principals (including the spy) do not know it, then X can authenticate that the message was sent by Y and is unmodified:

```
Rule4_1 : "Know Y (Encrypt M (Kpb X))
      ∧ Rcv X (Encrypt M (Kpb X))
      ∧ (∀Z. ((Z≠X ∧ Z≠Y)
      ⟶ ¬(Know Z M))) ⟹ Auth X Y M"
```

If another principal knows the message, then X will unauthenticate it:

```
Rule4_2 : "∀X Y. (∃Z. (Z≠X ∧ Z≠Y ∧ Know Z M))
      ⟹ ¬(Auth X Y M)"
```

The above notations, data structures, functions, predicates, assumptions and rules form the basic framework of the knowledge-based security protocol verification method.

2.5 Use of the Framework

With the implementation of the framework in Isabelle, we can now prove security properties and then prove the correctness of cryptographic protocols or find flaws in them mechanically. To achieve these, we need to conduct the following steps:

1. *Adjusting the notations and definitions in Isabelle.* Since cryptographic protocols may use different notations and definitions, we sometimes need to adjust them slightly according to the protocols we plan to verify.

2. *Modelling or formalising the protocols in Isabelle notation.* Before we can start to verify the protocols, we first need to describe them in the language which can be understood by the implementation of the framework and the Isabelle reasoning platform.

3. *Proving basic properties of the protocols.* Some basic properties of components of the framework and the protocols need to be proved. Most of the proofs of these properties can be reused in proving other protocols.

4. *Proving security properties of the protocols.* Further security properties should then be proved based on the basic properties. If all final security guarantees can be proved at this staged, the correctness of the protocols can be proved.

These steps can be illustrated as Figure 1.

The above steps are guidelines for proving the correctness of cryptographic protocols. If we encounter any problems in proving basic properties, security properties or security guarantees, these problems may imply that there are some flaws in the protocols, although they cannot prove the existence of the flaws. But the information given by the system could help us identify the flaws.

Now let us consider the Needham-Schroeder protocol, which is widely accepted as a "standard testbed" for security protocol formal verification methods, as an example.

3 OVERVIEW OF THE N-S PROTOCOL

In a vulnerable network environment with malicious spies and intruders, principals engaged in network communication cannot guarantee that who they are talking to is exactly the person they want to communicate with. *Authentication* is therefore used to help a principal verify the identity of the other one. In 1978, Needham and Schroeder proposed a protocol with the intent of providing mutual authentication (Needham, 1978). The protocol was considered correct for nearly two decades, until it was broken by Lowe using CSP with FDR as the model checker (Lowe, 1995, 1996).

The simplified Needham-Schroeder protocol can be described as follows [1]:

$$\begin{array}{lll} 1. & A \rightarrow B & : \{N_a, A\}_{K_b} \\ 2. & B \rightarrow A & : \{N_a, N_b\}_{K_a} \\ 3. & A \rightarrow B & : \{N_b\}_{K_b} \end{array}$$

[1] The original Needham-Schroeder protocol contains seven steps, four of which describe how A and B get the other's public encryption key from the server. In most formal verification models, principals' public key are considered open to the world (we made such assumptions in section 2.3). Therefore we ignore the four steps concerning public key distribution.

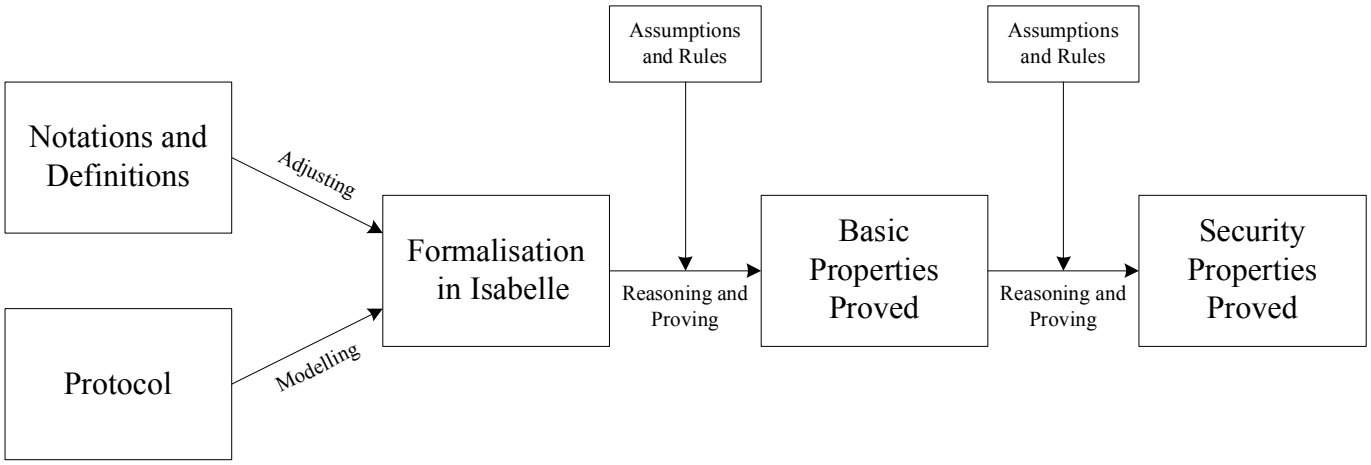

Figure 1: Steps to prove security protocols.

At the beginning, principal A composes a fresh nonce N_a and sends it to B with its own name, encrypted with B's public key. After B receives this message, B decrypts it and then reads the nonce N_a and knows who is seeking to communicate with it. B then sends back a message consisting of its own fresh nonce N_b as well as A's nonce N_a, encrypted with A's public key K_a, to A. To respond to B's message, A then returns B's nonce N_b to B.

Lowe fixed this protocol after he found a flaw in it by adding B's name into the second step (Lowe, 1995). The fixed protocol can be described as follows:

$$\begin{array}{lll} 1. \quad A \rightarrow B & : & \{N_a, A\}_{K_b} \\ 2. \quad B \rightarrow A & : & \{N_a, N_b, B\}_{K_a} \\ 3. \quad A \rightarrow B & : & \{N_b\}_{K_b} \end{array}$$

The aim of this protocol is to establish mutual authentication between an *initiator* A and a *responder* B. After step 2, A believes that B is responding to it because A sent N_a encrypted with B's public key to B and only B can decrypt it and read the content. Similarly, B is assured A's identity after step 3.

4 VERIFYING THE ORIGINAL PROTOCOL

To verify the original Needham-Schroeder protocol, we first need to model it in our framework, then prove a number of important lemmas and properties, and ultimately prove the final guarantees.

4.1 Modelling the Protocol

Normally, the protocol can be formalised as three steps for all honest principals:

1. First, principal A sends a compound message consisting of its nonce and name to principal B:

```
NS1 : "Send A B (Encrypt {Nonce (Nonce_of A),
       Principal A} (Kpb B))"
```

2. Second, if the first step has been successfully carried out, principal B will correspondingly send a compound message consisting of A's nonce, its own nonce and name to principal A:

```
NS2 : "Send A B (Encrypt {Nonce (Nonce_of A),
       Principal A} (Kpb B))
   ⟹ Send B A (Encrypt {Nonce (Nonce_of A),
       Nonce (Nonce_of B)} (Kpb A))"
```

3. Lastly, if the second step has been successfully carried out, principal A will correspondingly send B's nonce back to principal B:

```
NS3 : "Send B A (Encrypt {Nonce (Nonce_of A),
       Nonce (Nonce_of B} (Kpb A))
   ⟹ Send A B (Encrypt (Nonce (Nonce_of B))
       (Kpb B))"
```

These three steps are enough for honest principals. However, the spy does not necessarily obey these rules. As we stated in the assumptions previously, it may send out forged messages to other honest principals, i.e. it may send any messages it knows to any other principals as if they are according to the protocol:

axioms

```
Fake : "Know Spy X ⟹ Send Spy P X"
```

Since the spy can send out forged messages which seem to be valid protocol messages and fool honest principals, these honest principals may respond to the forged messages innocently. So we have another two extra rules:

axioms

```
NS2_response_to_Spy : "Send Spy B (Encrypt
     {Nonce (Nonce_of D), Principal F} (Kpb B))
  ⟹ Send B Spy (Encrypt {Nonce (Nonce_of D),
     Nonce (Nonce_of B)} (Kpb Spy))"


NS3_response_to_Spy : "Send Spy A (Encrypt
     {Nonce (Nonce_of D), Nonce (Nonce_of E)}
     (Kpb A))
  ⟹ Send A Spy (Encrypt (Nonce (Nonce_of E))
     (Kpb Spy))"
```

Additionally, since it is a two-part authentication protocol, only two honest principals are involved. Without losing generality, we name them as *Friend 1* and *Friend 2* corresponding to A and B, respectively. If a principal does not equal to either of them, it must be the spy, since an honest principal will not involve itself in a protocol session which he should not take part in.

axioms

```
other_principal : "⟦ X ≠ Friend 1; X ≠ Friend 2 ⟧
     ⟹ X = Spy"
```

We can use all above assumptions, rules and the formalisation to verify the Needham-Schroeder public key protocol. All the following lemmas and goal are inferred from them. We denote them as a whole as Σ. Therefore, any lemma or goal F should be understood as $\Sigma \models F$. In the Isabelle implementation, the Σ is not written explicitly. However, it always exists in the whole proof.

4.2 Some Important Lemmas

To prove the final guarantees for the two participating principals, we need to first prove some lemmas. Most of them are straightforward. For example, if principal A sends to B a message encrypted by B's public encryption key, B will be able to read the content of this message.

Besides these straightforward properties, there are a number of important ones worth noting. One of these lemmas is that if a principal knows a message M, and M_1 is a part of M, then it should know the message M_1 as well:

lemma *know_part_imply [simp] :*

```
"Know A M ∧ msg_part M1 M ⟶ Know A M1"
```

Due to the inductive definition of the data type *message*, we need to prove this lemma by induction. Seven subgoals have been produced when we induct on the structure of message *M* by using the induction rule *induct_tac*. The subgoals concerning principal, nonce, key and encrypted massages, signatures and hashed messages can be simply proved by using the implication introduction (*impI*), conjunction elimination (*conjE*), conjunction introduction (*conjI*), disjunction elimination (*disjE*) and substitution (*ssubst*) rules provided by higher order logic (Nipkow, 2003). However, the subgoal concerning compound messages is a little more complex. In this case, we need to decompose all compound parts into pieces, and then prove them one by one.

In the proof, different rules are used. Besides normal higher-order rules, we also use the *elimination rule* (*erule*) which eliminates premise of no interest and the *destruction rule* (*drule*) which takes apart and destroys a premise. Besides that, the *assumption* rule, which means that the result we need to prove exists in assumptions, is also used. The "+" sign appearing after the *assumption* rule (and other rules) indicates that the current rule can be executed more than once whenever possible.

In addition, we also use several "atom lemmas", which state that a certain object is not dividable. These lemmas include *know_atom_principal*, *know_atom_nonce*, *know_atom_key*, *know_atom_encrypt*, *know_atom_sign* and *know_atom_hash*. Since the objects are defined as the simplest block of type *message*, the proof of these lemmas are trivial.

The proving steps of the lemma *know_part_imply* are as follows:

```
apply (induct_tac M)

apply (rule impI know_atom_principal)+
apply (erule conjE)
apply (rule conjI)
apply assumption+

apply (rule impI  know_atom_nonce)+
apply (erule conjE)
apply (rule conjI)
apply assumption+

apply (rule impI know_atom_key)+
apply (erule conjE)
apply (rule conjI)
apply assumption+

apply (rule impI)
apply (erule conjE)
apply (drule mpair_either)
apply (erule disjE)
apply (erule ssubst)
apply assumption
apply (drule Rule2_2)
apply assumption+
apply (rule impI know_atom_encrypt)+
apply (erule conjE)
apply (rule conjI)
apply assumption+

apply (rule impI know_atom_sign)+
apply (erule conjE)
apply (rule conjI)
apply assumption+

apply (rule impI know_atom_hash)+
apply (erule conjE)
apply (rule conjI)
by assumption+
```

With the lemma *know_part_imply*, it is easy for us to

prove the lemma *know_encrypted_part* which is a simplification rule. In the proof of this lemma, we use the *simp only* rule, which means that the system only uses the certain rule instead of searching all applicable simplification rules automatically. In addition, the *know_part* rule is the implication form of *know_part_imply*. And the *inv_publicKey* rule means that in public cryptosystem, a pair of public key and private key are reverse keys of each other.

```
lemma know_encrypted_part [simp] :
        "⟦ Know B (Encrypt M (Kpb B));
          msg_part M1 M ⟧ ⟹ Know B M1"

apply (rule know_part)
apply (rule conjI)
prefer 2
apply assumption
apply (rule Rule1_2)
apply (rule conjI)
apply assumption
by (simp only: inv_publicKey KeyAssump2)+
```

Another important lemma states that if a principal sends to another principal a compound message encrypted with the second principal's public encryption key, the latter will eventually know the parts of it:

```
lemma know_send_encrypted_part [simp] :
        "⟦ Send A B (Encrypt M (Kpb B));
         msg_part M1 M ⟧ ⟹ Know B M1"
```

This lemma can by easily proved using the lemma *know_encrypted_part* together with rules 3_3 and 3_1 of our framework. To make proofs clear, we usually prove simplest subgoals first. So when we find the second subgoal generated by the system is simpler than the first subgoal, we may use the command *prefer 2* to move the second subgoal to the front.

```
apply (rule know_encrypted_part)
prefer 2
apply assumption
apply (rule Rule3_3)
apply (rule Rule3_1)
by assumption
```

In doing so, it simplifies the final proof greatly.

4.3 Secrecy of Nonces

The correctness of the Needham-Schroeder protocol relies greatly on the secrecy of the nonces used by A and B (Paulson, 1997b). So the key point for proving the Needham-Schroeder protocol is to prove the secrecy of nonces of A and B.

Our proofs base these nonce secrecy lemmas on such an assumption: if the spy knows the contents of the nonce of an honest principal, then the nonce should have been transmitted over the network in plain text or encrypted using an encryption key whose corresponding decryption key is known by the spy.

```
axioms spy_know_encrypted_nonce :
      "Know Spy (Encrypt (Nonce (Nonce_of
        (Friend n))) (Kpb (Friend n)))
      ⟹ Send C Spy (Encrypt (Nonce (Nonce_of
        (Friend n))) (Kpb Spy))"
```

The first secret property we need to prove is that the spy can never know B's nonce, which can be described as follows:

```
lemma spy_not_know_nonce_2 :
   "¬ Know Spy (Nonce (Nonce_of (Friend 2)))"
```

We prove this lemma by the "negative approach", i.e., if the spy knows B's nonce, it may send to B this nonce encrypted by B's public key, which does not accord to the protocol.

In the proof of the property, we have to prove the following subgoal:

```
    Send ?C6 Spy (Encrypt (Nonce (Nonce_of
      (Friend 2))) (Kpb Spy))
⟹ Spy = Friend 2
```

where the *?C6* is a temporary variable generated by the system automatically. The name of the temporary variable is unimportant. This subgoal means that if someone sends B's nonce to the spy using the spy's public key, then the spy and B is actually the same principal. We encounter difficulties when we want to prove this subgoal, even with the help of Isabelle's automatic tactics search function. This naturally makes us think that the spy may impersonate another principal and illegally get B's nonce. Since only step 3 of the protocol has such form, we may assume that A sends B's nonce to the spy. Then we can easily backtrack along the protocol and find the following attack which was first found by Lowe (Lowe, 1995):

Session 1, Step 1	$A \rightarrow I$	:	$\{N_a, A\}_{K_i}$
Session 2, Step 1	$I(A) \rightarrow B$	:	$\{N_a, A\}_{K_b}$
Session 2, Step 2	$B \rightarrow I(A)$	:	$\{N_a, N_b\}_{K_a}$
Session 1, Step 2	$I \rightarrow A$	:	$\{N_a, N_b\}_{K_a}$
Session 1, Step 3	$A \rightarrow I$	:	$\{N_b\}_{K_i}$
Session 2, Step 3	$I(A) \rightarrow B$	:	$\{N_b\}_{K_b}$

Actually, we can formally prove the following lemma which indicates that the spy can know B's nonce before we know the attack:

```
lemma spy_know_nonce_2 :
       "Know Spy (Nonce (Nonce_of (Friend 2)))"
apply (rule Rule1_2)
apply (rule conjI)
apply (rule Rule3_3)
apply (rule Rule3_1)
apply (rule NS3_response_to_Spy)
apply (rule Fake)
apply (rule Rule3_3)
```

```
apply (rule Rule3_1)
apply (rule NS2_response_to_Spy)
apply (rule Fake)
apply (rule Rule1_1)
apply (rule conjI)
apply (rule Rule2_1)
apply (rule conjI)
apply (rule conjunct1)
apply (rule Rule2_2)
apply (rule Rule1_2)
apply (rule conjI)
apply (rule Rule3_3)
apply (rule Rule3_1)
apply (rule NS1)
by (simp only: inv_publicKey KeyAssump1 KeyAssump2
    KnowName)+
```

With the lemma *spy_know_nonce_2*, we can easily prove the following goal indicating that B cannot authenticate the originality of the third message with the help of rule *4_2*. In the proof, we use the *auto* rule which means that the current subgoal is straightforward and can be automatically solved by Isabelle. We also use the existence introduction rule *exI*.

```
goal B_not_trust_NS3 :
      "¬ Auth (Friend 2) (Friend 1)
       (Nonce (Nonce_of (Friend 2)))"
apply (rule Rule4_2)
apply (rule exI)
apply (rule conjI)
prefer 2
apply (rule conjI)
prefer 2
apply (rule spy_know_nonce_2)
by auto
```

5 PROVING THE FIXED PROTOCOL

Lowe added B's name into the second step of the Needham-Schroeder protocol to fix it, therefore we need to reformalise the second and third steps accordingly:

```
NS2_Fix : "Send A B (Encrypt {Nonce (Nonce_of A),
          Principal A} (Kpb B))
          ⟹ Send B A (Encrypt {Nonce (Nonce_of A),
          Nonce (Nonce_of B),
          Principal B} (Kpb A))"

NS3_Fix : "Send B A (Encrypt {Nonce (Nonce_of A),
          Nonce (Nonce_of B),
          Principal B} (Kpb A))
          ⟹ Send A B (Encrypt (Nonce (Nonce_of B))
          (Kpb B))"
```

Meanwhile, we can use the added information to decide whether the second message in protocol session is a faked one or not. If an honest principal sends out a message with the form of the message in the *NS2_Fix*, then the third component of the message should be the name of the sender, otherwise the receiver should decline the message and terminate the current protocol session:

```
axioms
      NS2_decline : "Send B A (Encrypt {Nonce
         (Nonce_of D), Nonce (Nonce_of E),
         Principal C} (Kpb A))
         ⟹ B = C"
```

5.1 Secrecy of Nonces

With the above preparation we can prove the *spy_not_know_nonce_2*, which was failed in the original protocol. In the proof we use contrapositive rule with the following form:

```
apply (rule_tac Q = "Formula" in contrapos_nn)
```

where *Formula* is the result we infer when we suppose the conclusion is false. The proof of the lemma *spy_not_know_nonce_2* is as follows:

```
lemma
     spy_not_know_nonce_2 :
       "¬ Know Spy (Nonce (Nonce_of (Friend 2)))"
apply (rule_tac Q =
       "Send Spy (Friend 2)
        (Encrypt (Nonce (Nonce_of (Friend 2)))
        (Kpb (Friend 2)))" in contrapos_nn)
prefer 2
apply (rule Fake)
apply (rule Rule1_1)
apply (rule conjI)
apply assumption
apply (simp only: KeyAssump1)
apply (rule_tac Q = "Spy = Friend 2" in contrapos_nn)
prefer 2
apply (drule Rule3_4)
apply (drule spy_know_encrypted_nonce)
apply (rule NS2_decline)
apply (rule NS3_response_to_Spy)
apply assumption
apply (simp only: Honest_not_Spy)
by auto
```

Another lemma *spy_not_know_N1N2* indicates that the spy can never see the contents of the message containing A and B's nonces and B's name in step 2.

```
lemma Spy_not_know_N1N2 :
   "¬ Know Spy {Nonce (Nonce_of (Friend 1)),
      Nonce (Nonce_of (Friend 2)),
      Principal (Friend 2)}"
```

This lemma can be easily inferred from the above lemmas, since if the spy knows the compound message in step 2, it will know B's nonce.

5.2 Proving Guarantee for B

The guarantee for B after step 3 is that B authenticates that A really has sent B's nonce (encrypted by B's public encryption key) to B and this message has not been modified by the spy:

```
theorem B_trust_NS3 : "Auth (Friend 2)
   (Friend 1) (Nonce (Nonce_of (Friend 2)))"
```

With all above lemmas and properties, the proof is not difficult. Rule 4_1 is applied first. It produces three subgoals:

- A knows B's nonce encrypted by B's public key;
- B has received such a message;
- No other principal knows B's nonce.

For the first subgoal, since principal B has sent its nonce to A (together with B's own name and A's nonce, and encrypted by A's public encryption key), A must have got and read the content of it. A can encrypt it using B's public encryption key. Step 3 of the protocol guarantees the second subgoal. The third subgoal is shown by the lemma *spy_not_know_nonce_2*. With all the subgoals resolved, the guarantee for B is proved.

5.3 Proving Guarantee for A

Correspondingly, the guarantee for A after step 2 is that A authenticates that B really has sent A's nonce, B's nonce and B's name (encrypted by A's public encryption key) to A and this message has not been modified by the spy:

```
theorem A_trust_NS2 : "Auth (Friend 1)
   (Friend 2) {(Nonce (Nonce_of (Friend 1))),
   (Nonce (Nonce_of (Friend 2))),
   (Principal (Friend 2))}"
```

Again, rule 4_1 is applied. Another three subgoals are produced:

- B knows the compound message consisting of A's nonce, B's nonce and B's name encrypted by A's public encryption key;
- A has received such a compound message;
- No other principal knows such a compound message.

When step 1 finishes, B has received A's nonce. B also knows B's nonce and name. B can then encrypt it using A's public encryption key. Step 2 of the protocol guarantees that A receives the message. The third subgoal has been shown by the lemma *spy_not_know_N1N2*. Therefore the guarantee for A is proved.

6 CONCLUSIONS

In this paper, we have presented a new framework to prove the correctness of security protocols. We model the protocols into logic formulae and infer them by analysing principals' knowledge states — what they can know and what they can never know. We have introduced the notations, data structures, functions, predicates, assumptions and inference rules to describe the knowledge of principals and the relationships among them. The rules give the conditions under which the knowledge can be changed, and how they can be changed.

We implement our framework using Isabelle. We have implemented all the data structures, functions, predicates, assumptions and inference rules in Isabelle. With this implementation, we are able to prove the correctness of protocols or find flaws mechanically using our framework — modelling the protocol and decomposing the goal, proving the necessary lemmas and properties, and proving the final guarantees. All the proving details are generated by Isabelle, thereby saving users a significant amount of time. Since we do not need to do state space searching, our framework is more efficient than state based methods. In addition, our framework also takes the cases concerning multiple interleaving sessions into consideration, making the method feasible in more areas than rule-based methods.

To show the effectiveness of our framework, we have used the Needham-Schroeder public key authentication protocol and Lowe's fix as an example. The example shows how to use the framework and its implementation to find flaws in the original protocol and prove the correctness of the fixed protocol.

Rule based methods generally overlook the attacks involving interleaving sessions. In verifying the Needham-Schroeder public key authentication protocols, The BAN logic (Burrows, 1990), Bolignano's model (Bolignano, 1996) and Chen *et al.*'s ENDL framework (Chen, 2004) fail to find Lowe's attack since they do not consider the situation that two or more sessions are running at the same time, although they successfully prove some other security properties. For example, in the verification of BAN logic, the protocol steps should first be changed to idealised form, and assumptions about the initial state should be written. After that, the protocol should be annotated in the way that logical formulae are attached to statements of the protocol, as assertions about the state of the system after each statement. Then the logical postulates are applied to the assumptions and the assertions to derive the beliefs held by the participants of the protocols. This procedure may be repeated as new assumptions are found to be necessary and as the idealised protocol is refined. Although this method is concise and elegant, the modeling of freshness is problematic. As in most modal logics it is in particular not possible to distinguish between freshness of creation and freshness of receipt. Therefore it is possible for the spy to get the nonce in one session and then reuse it in another without being detected. Most state based methods

can find the attack quickly. Our method finds the attack in eight seconds with the help of Isabelle on a computer with a Pentium IV CPU, 256M memory and a 40G hard disk, the similar speed as state based methods.

Although state based methods work well in the simple Needham-Schroeder public key protocols, larger protocols are quite difficult to them. The state spaces become extreme huge when protocols become more complicated. For example, in the NRL Protocol Analyzer, actions of legitimate participants are specified by the user as state transitions. Input to the transitions are values of local state variables and messages received by the participant, the latter assumed to have been generated or passed on by the intruder, and outputs are the new values of local state variables, and messages sent by the participant, which are subject to interception and modification by the intruder. Since for each input it is possible to have a number of different outputs, when the number of steps grows, the number of states will increase dramatically. Our method spends about one minute to verify the secure electronic transactions (SET) protocol (Ma, 2005b), while state based methods keep running in reasonable waiting time. More theoretical analysis of efficiency is beyond this paper and will be left to future work.

In addition, we also tried other three protocols, namely the IEEE 802.11 wireless authentication protocol, the password authentication protocol (PAP), and the Secure Electronic Transactions (SET) protocol. Only the CSP method found the attack in the PAP protocol (Kim, 2004), but the patch it provided was also insecure. With our method, we successfully found flaws in the original protocol as well as in the protocol with the patch. We also proposed a new patch and proved the correctness of the new patch (Ma, 2006a). Besides that, we found a subtle flaw in the IEEE 802.11 wireless authentication protocol, provided a fix to it, and proved the correctness of the fix (Ma, 2006b). No other method has been reported to find any flaw in the same protocol. For the SET protocol, state based methods tend to keep running for very long time. For rule based method, only the inductive method reported to have verified it. However, the inductive method spent more then 300 seconds (Bella, 2005) in a single step "purchase", while we spend less than 60 seconds to verify all five steps.

Besides all above advantages, when using our method, most of the code for one protocol can be reused for other protocols, making proving other protocols easily and quickly.

ACKNOWLEDGMENT

Our research has been supported by EC, EPSRC, the National Natural Science Foundation of China, and Hong Kong K C Wong Education Foundation.

REFERENCES

Bella, G., Massacci, F., and Paulson, L. (2005). 'An overview of the verification of SET'. *International Journal of Information Security*, Vol. 4, No. 1-2, pp.17–28.

Bolignano, D. (1996). 'An approach to the formal verification of cryptographic protocols'. *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, New Delhi, India, pp.106–118.

Burrows, M., Abadi, M., and Needham, R. (1990). 'A logic of authentication'. *ACM Transactions on Computer Systems*, Vol. 8, No. 1, pp.18–36.

Chen, Q. (2004). 'The verification logic for secure transaction protocols'. PhD thesis, University of Technology, Sydney, Australia.

Chen, Q., Zhang, C., and Zhang, S. (2005). 'A logical framework ENDL for verifying secure transaction protocols'. *Knowledge and Information Systems*, Vol. 7, No. 1, pp.84–109.

Cheng, X., Ma, X., Cheng, M., and Huang, S. C.-H. (2005). 'Proving secure properties of cryptographic protocols'. *Proceedings of the 24th IEEE International Performance Computing and Communications Conference (IPCCC 2005)*, Phoenix, Arizona, USA, pp.3–9.

Kim, I., and Choi, J. (2004). 'Formal verification of PAP and EAP-MD5 protocols in wireless networks: FDR model checking'. *Proceedings of the 18th International Conference on Advanced Information Networking and Applications*, Fukuoka, Japan, pp.264–269.

Liebl, A. (1993). 'Authentication in distributed systems: A bibliography'. *ACM SIGOPS Operating Systems Review*, Vol. 27, No. 4, pp.122–136.

Lowe, G. (1995). 'An attack on the Needham-Schroeder public-key authentication protocol'. *Information Processing Letters*, Vol. 56, No. 3, pp.131–133.

Lowe, G. (1996). 'Breaking and fixing the Needham-Schroeder public-key protocol using FDR'. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, Springer Verlag, pp.147–166.

Ma, X., Cheng, X., and McCrindle, R. (2005a). 'Knowledge based approach for mechanically verifying security protocols'. *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, Edinburgh, Scotland, UK, pp.1572–1573.

Ma, X., and Cheng, X. (2005b). 'Formal verification of Merchant Registration phase of SET protocol'. *Proceedings of the 11th Annual Conference of Chinese Automation and Computing Society in UK*, Sheffield, UK.

Ma, X., McCrindle, R., and Cheng, X. (2006a). 'Verifying an enhanced version of PAP protocol in wireless communications'. *Proceedings of the 2nd ACIS International Workshop on Self-Assembling Wireless Networks*, Las Vegas, Nevada, USA.

Ma, X., Cheng, X., and McCrindle, R. (2006b). 'Breaking and fixing IEEE 802.11 wireless authentication protocol', *IEE Proceedings — Information Security*, under review.

Meadows, C. A. (1996a). 'Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches'. In Bertino, E., Kurth, H., Martella, G., and Montolivo, E., editors, *Computer Security ESORICS 96*, volume 1146 of *Lecture Notes in Computer Science*, Springer Verlag, pp.351–364.

Meadows, C. A. (1996b). 'The NRL protocol analyzer: An overview'. *Journal of Logic Programming*, Vol. 26, No. 2, pp.113–131.

Needham, R. and Schroeder, M. (1978). 'Using encryption for authentication in large networks of computers'. *Communications of the ACM*, Vol. 21, No. 12, pp.993–999.

Nipkow, T., Paulson, L. C., and Wenzel, M. (2003). *Isabelle/HOL: A proof assistant for higher-order logic*. Springer Verlag, Heidelberg.

Paulson, L. C. (1997a). 'Proving properties of security protocols by induction'. *Proceedings of the 10th Computer Security Foundations Workshop*, Rockport, Massachusetts, pp.70–83.

Paulson, L. C. (1997b). 'Mechanized proofs of security protocols: Needham-Schroeder with public keys'. Technical Report 413, Computer Laboratory, University of Cambridge.

Paulson, L. C. (1998). 'The inductive approach to verifying cryptographic protocols'. *Journal of Computer Security*, Vol. 6, No. 1-2, pp.85–128.