A New Algorithm for the Automatic Verification of Authentication Protocols: From Specifications to Flaws and Attack Scenarios

M. Debbabi M. Mejri N. Tawbi I. Yahmadi

Computer Science Department,
Laval University, Quebec, G1K 7P4,
Canada.
{debabi,mejri,tawbi,yahmadi}@ift.ulaval.ca

Abstract

We present a new formal automatic approach to the verification of authentication protocols. Our method could be applied without any prior specification of properties or invariant. It only needs the protocol specification from which it generates, if any, the set of possible flaws as well as the corresponding attack scenarios. This approach consists of three steps. First, the extraction of roles from the protocol specification. Second, the generation of a proof system that models the intruder abilities to perform communications and computations from the protocol specification. In addition to the classical known intruder computational abilities such as encryption and decryption, we also consider the computations that result from the possible instrumentations of the protocol. Third, the verification is performed according to the extracted roles as well as to the deductive system. This verification consists in checking whether the intruder can answer all the challenges uttered by a particular role. If it is the case, an attack scenario is automatically constructed. To exemplify the usefulness and efficiency of our approach, we illustrate it on Woo and Lam authentication protocol. Abadi and Needham have shown that the protocol is insecure and they proposed a new corrected version. In this paper, we present new unknown flaws in the Woo and Lam protocol as well as in its corrected version.

Keywords: Algorithm, Automatic Formal Verification, Cryptographic Protocols, Authentication, Proof System, Flaws, Attack Scenario.

1 Motivations and Background

In secure distributed systems it is essential that principals (persons, hosts, computers, etc.) could prove their identities to each other. The mechanism of proving the identity over networked systems is called authentication. Typically, cryptographic protocols are used to ensure authentication and related purposes. A cryptographic protocol is a precisely defined sequence of communication and computation steps that use cryptographic mechanisms such as message encryption and decryption. We assume that the reader is familiar with cryptography basics and the associated concepts.

It is well known that the design of authentication cryptographic protocols is error prone. Several protocols have been shown flawed in computer security literature. Consequently, a surge of interest has been expressed in the development of formal methods and tools for the specification, design and analysis of cryptographic protocols. A complete bibliography and a comparative study of these methods can be found in [13, 31, 30, 36, 39, 24, 43, 44, 14]. These methods can be classified as follows: logical methods, general purpose formal methods and process algebra methods.

Typically, logical methods rest on the use of modal (epistemic, temporal and/or doxatic) logics. The logic is used to specify the protocol (idealization) as well as the security properties. In 1989, Burrows, Abadi and Needham devised BAN, a modal logic of belief for the specification and verification of cryptographic protocols [11, 12]. BAN is the most known and famous logic dedicated to cryptographic protocols. Since then, plenty of derived logics have been advanced [1, 20, 15, 29, 19]. In 1990, Bieber [5] developed

CKT5, a modal logic of knowledge which has been revised and extended by Carlsen in [13] and Snekkenes in [42]. Concurrently, many other logics attempted to combine several aspects of modal logic such as belief, knowledge and trust [18, 35, 37, 45]. These methods have been successfully used to detect many flaws in cryptographic protocols and they are very expressive while specifying security properties. Nevertheless, they are not very suitable to specify the protocols themselves. In fact, the protocols are often translated into a set of logical formulas. The translation process, often referred to as idealization, is error prone since it aims to translate an operational description into a logical one. Furthermore, the idealization is not systematic. Moreover, most of the proposed logics while proved sound with respect to some semantics are generally incomplete. In addition, the verification of the protocol is always manual and semi-formal.

Another trend in formal cryptographic development consists in the accommodation of some well known general purpose formal methods. Representative specification languages that have been used in such accommodations are LOTOS [48, 49, 47, 10], B, VDM [7, 46, 6], HOL [42], Ina Jo [22, 23], Z [9, 41] and Coq [8]. Although these formal methods are now firmly established and known to be of great use in specification and verification, it remains that these methods are not dedicated to cryptographic protocols. In addition, these methods need much of expert assistance during the verification process. Actually, they rely on manual or interactive theorem proving techniques.

Lately, the use of process algebra for cryptographic protocol specification and verification has been explored. In 1995, Gavin Lowe [25, 28, 27, 26] was the first to use CSP [21] and model-checking techniques for cryptographic protocol analysis. The protocol is specified as a set of communicating sequential processes that are running in parallel and interacting with their surroundings. The verification is performed by extracting a model (usually a finite state transition system) from the specification and checking that model against a logical specification (a formula over a modal temporal logic) or a behavioral specification (a process term). A similar approach was developed by Bill Roscoe, Paul Gardiner, Dave Jackson and Janson Hulance in [38, 16, 17] and Steve Schneider in [40]. Recently, Abadi and Gordon [2, 3] advanced Spi, a calculus for cryptographic protocols. Spi is built on top of the π-calculus [32, 33, 34], a mobile process algebra. It has been devised for the description and analysis of security protocols. The process algebra-based methods have been successfully used in the detection of several flaws in well known cryptographic protocols. The approach seems to be very promising and useful. However, it is well known that the underlying verification techniques, mainly those based on model-checking are problematic in the presence of processes that exhibit infinite behaviors. Accordingly, the infinite aspects of cryptographic protocols are usually not supported in the verification process. Notice also that the specification of security properties in terms of process agents or modal formulae is neither straightforward nor systematic (except in the case of Spi).

In this paper, we present a new formal approach for the analysis of authentication cryptographic protocols. This approach is fully automatic, formal and does not necessitate any specification of any protocol property or invariant. The analysis of a given cryptographic protocol is structured in three main steps:

- First, starting from a classical protocol description, we extract what we call protocol roles. Actually, a role is a protocol abstraction where the emphasis is put on only one principal. Such an abstraction allows us to point out, for each principal, his interpretation of the exchanged protocol messages.
- Second, the intruder abilities are automatically extracted. Apart from the well known abilities such as the complete control of the network, encryption and decryption, we consider all the computational abilities provided by the protocol itself. Generally, the intruder can initiate one or more sessions of the protocol so as to get or reconstruct a particular information to be used into an attack. In our approach, such computation abilities are captured as a finite proof system in which the inference is based on rewriting modulo syntactic unification.
- Third, the roles together with the proof system are combined to perform the protocol verification. The latter consists in checking whether one can instrument a particular role by answering all the challenges uttered by this role. The answers are merely theorems established in the deductive proof system.

The main contributions of this work are:

- A new approach for the analysis of cryptographic protocols. This approach is formal, fully automatic and does not necessitate any specification of any protocol property or invariant. It takes as a parameter the protocol specification and generates the set of flaws, if any, as well as the corresponding attack scenarios.

- An illustration of the usefulness and efficiency of our approach, on the analysis of the Woo and Lam authentication protocol. Abadi and Needham have shown that the protocol is insecure and they proposed a new corrected version. In this paper, we discover new unknown flaws in the Woo and Lam protocol and in the corrected version of Abadi and Needham.

We consider here a subclass of authentication cryptographic protocols. The approach deals with protocols that use symmetric-key cryptography, at most one server and where the only exchanged messages are constructed over nonces, principal identities and symmetric-keys. Moreover, we assume that principals recognize the use of their keys.

The rest of this paper is structured as follows: In Section 2 we recall briefly some basic notions of cryptographic protocols. Section 3 is devoted to an informal presentation of the verification algorithm. In Section 4, we present the formal description of the algorithm and we show how the proof system is generated. In Section 5, we illustrate our approach on the analysis of the corrected version of the Woo and Lam authentication protocol. A few concluding remarks and a discussion of further research are ultimately sketched as a conclusion in Section 6.

2 Basics

As defined in [50, 51], a protocol is a precisely defined sequence of communication and computation steps. A communication step transfers messages from one principal (sender) to another (receiver), while a computation step updates a principal's internal state. A protocol with a security objective is called a cryptographic protocol. Cryptographic functions are used to achieve such an objective. Authentication protocols are a special kind of cryptographic protocols where the primary aim is to allow principals to identify themselves to each other.

In the sequel, the statement $A \longrightarrow B : M$ will be used to denote the transmission of a message M from the principal A to the principal B. A message is composed of one or several primitive words. A message encrypted with key k is written $\{M\}_k$ and forms a word by itself. Concatenated messages are separated by commas. Message contents (words) follow naming conventions: Encryption keys and nonces are respectively written k and N. Principals are written A, B, S and I, where A and B stand for principals, S for a server and I for a potential intruder. Subscripts will be used to denote principal association, thus for example N_a is a nonce that belongs to A and k_{as} is a shared key between A and S.

In the sequel, we recall the one-way Woo and Lam authentication protocol as presented in [50, 51]. This protocol relies on symmetric-key cryptography. The protocol runs as follows when the principal A wants to prove his identity to the principal B:

Here N_b is a nonce, a random number generated by B specially for this protocol run, S is a server and k_{as} and k_{bs} are keys that A and B initially share with S. The description of the protocol can be read as follows: (1) A initiates the protocol and claims his identity to B; (2) B replies by sending the nonce N_b and asking A to encrypt it under k_{as} in order to prove what he claimed; (3) A returns the nonce N_b encrypted under k_{as}; (4) B forwards the response encrypted, together with A's identity, under k_{bs} for verification; (5) S decrypts the received message using B's key, extracts the encrypted component and decrypts it using A's key and re-encrypts under B's key. If S replies $\{N_b\}_{k_{bs}}$, then B will find N_b after decrypting it and he should be convinced that A is really running this session with him. The protocol is given in Table 1.

Woo and Lam stated that the protocol is correct if: "whenever a responder finishes the execution of the protocol, the initiator is in fact the principal claimed in the initial message". Nevertheless, the protocol is known to be flawed. In particular, Abadi and Needham presented an attack to the protocol in [4]. They noticed that an attack is possible because there is no connection between B's request and S's reply. The authors discussed again the protocol in their paper and finally suggested the new corrected version given in Table 2.

The next section is devoted to an informal presentation of our verification algorithm. The latter will be illustrated over the Woo and Lam authentication protocol described above.

3 Informal Presentation

Starting from a protocol description, the algorithm operates in three main steps:

(a) Role extraction
(b) Proof system generation
(c) Verification

Message 1.	A	$\longrightarrow$	B	:	A
Message 2.	B	$\longrightarrow$	A	:	N_b
Message 3.	A	$\longrightarrow$	B	:	$\{N_b\}_{k_{as}}$
Message 4.	B	$\longrightarrow$	S	:	$\{A, \{N_b\}_{k_{as}}\}_{k_{bs}}$
Message 5.	S	$\longrightarrow$	B	:	$\{N_b\}_{k_{bs}}$

Table 1: The Woo and Lam Authentication Protocol

Message 1.	A	$\longrightarrow$	B	:	A
Message 2.	B	$\longrightarrow$	A	:	N_b
Message 3.	A	$\longrightarrow$	B	:	$\{N_b\}_{k_{as}}$
Message 4.	B	$\longrightarrow$	S	:	$A, B, \{N_b\}_{k_{as}}$
Message 5.	S	$\longrightarrow$	B	:	$\{A, N_b\}_{k_{bs}}$

Table 2: The Corrected Version of the Woo and Lam Authentication Protocol

In the sequel, we will motivate, explain and illustrate each of these three steps. The illustration will be carried out on the Woo and Lam authentication protocol.

3.1 Role Extraction

Roles are protocol abstractions where the emphasis is put, for a specific role, on a particular principal. A role reflects the way some principal perceives the protocol messages. To understand the difference between roles and principals, let us take the following simple example: suppose that we have two principals A_1 and A_2 and suppose that the principal A_1 wishes to prove his identity to A_2, in this case the principal A_1 has to play the role A and the principal A_2 has to play the role B. However, if, later, the principal A_2 wishes to prove his identity to A_1, A_2 must play the A's role and A_1 the B's one. The reader should notice that throughout the rest of this paper roles and principals will be confused as far as ambiguity could be avoided.

For instance, in the case of the Woo and Lam protocol of Table 1, three roles could be extracted: A, B and S. The principal, playing the role A, participates in the protocol through three main steps: First, A sends his identity to the principal B. Second, he receives a nonce N_b from B. Third, he sends the message $\{N_b\}_{k_{as}}$ to B. Hence, the role associated to A could be written as the following sequence of actions:

$$Role(A) = \langle !, A, B\rangle\langle ?, N_b, B\rangle\langle !, \{N_b\}_{k_{as}}, B\rangle$$

An action is a triple of the form $\langle dir, m, P\rangle$ where dir is a direction symbol (either ? meaning input or ! meaning output), m is a message and P is a principal identifier.

The principal playing role B participates in the protocol of Table 1 through five actions. First, he receives a principal identifier, say A. Second, he generates a fresh nonce, say N_b, and sends it to the agent A. Third, he receives from A the message $\{N_b\}_{k_{as}}$. Fourth, he sends to the server the message $\{A, \{N_b\}_{k_{as}}\}_{k_{bs}}$. Fifth, he receives from the server the message $\{N_b\}_{k_{bs}}$. Accordingly, the corresponding role is:

$$\begin{array}{rcl} Role(B) & = & \langle ?, A, A\rangle \\ & & \langle !, N_b, A\rangle \\ & & \langle ?, \{N_b\}_{k_{as}}, A\rangle \\ & & \langle !, \{A, \{N_b\}_{k_{as}}\}_{k_{bs}}, S\rangle \\ & & \langle ?, \{N_b\}_{k_{bs}}, S\rangle \end{array}$$

The principal playing the role of the server S participates in the protocol through two actions. First, he receives from B the message $\{A, \{N_b\}_{k_{as}}\}_{k_{bs}}$. Second, he sends to B the message $\{N_b\}_{k_{bs}}$. Hence, the role of S is given by:

$$Role(S) = \langle ?, \{A, \{N_b\}_{k_{as}}\}_{k_{bs}}, B\rangle\langle !, \{N_b\}_{k_{bs}}, B\rangle$$

At that point, we are ready to present the second step of our algorithm i.e. the proof system generation.

3.2 Proof System Generation

In cryptographic protocol design, we always assume the presence of an intruder that has a complete control over the communication network. Accordingly, the intruder is able to intercept, modify, store, retrieve and send any circulating message in the network. Moreover, we assume that the intruder has the usual encryption and decryption abilities. In this work, we consider another valuable source of information available to the intruder: the protocol itself. Actually, the protocol may be viewed by the intruder as a computation engine which may be used to synthesize and deliver some particular information that makes possible a fatal attack.

In our approach, we capture all these intruder abilities by a deductive proof system. We associate to each protocol step an inference rule. Each inference rule captures one possible instrumentation of the protocol. The general form of an inference rule is:

$$\frac{p_1 \ldots p_n}{m}\ c$$

where $p_1, \ldots, p_n, m$ are messages and c is a boolean formula. This inference rule should be read as follows: the intruder could supply the protocol with the messages $p1, \ldots, p_n$ and get the message m from the protocol provided that the side condition c holds[1]. In fact, the side condition refers to those freshness conditions dictated by the protocol. Furthermore, we will endow each inference rule with a sequence of protocol steps (a scenario) showing *how* the intruder could instrument the protocol with the information $p_1, \ldots, p_n$ so as to get the message m.

Now, let us see how this applies to the Woo and Lam protocol of Table 1. For the sake of clarity, we present here the inference rules together with the associated scenarios without explaining how they are generated. Such explanations will be given further in the sequel.

The first step in the Woo and Lam protocol of Table 1 is:

$$1.\ A \longrightarrow B : A$$

The inference rule associated with this protocol step is:

$$\frac{}{A}$$

meaning that the intruder could get from the protocol the identity of any principal wishing to initiate a protocol session. Here is the scenario that makes this possible:

$$1.\ A \longrightarrow I(B) : A$$

This sequence stipulates that the intruder could get the identity of a principal A by intercepting the message sent by this principal when wishing to initiate a new protocol session. In doing so, the intruder I performs a masquerade on the role of a principal B, which is written $I(B)$. Later in this section, we will present inference rules that capture sophisticated computations.

The second step in the Woo and Lam protocol of Table 1 is:

$$2.\ B \longrightarrow A : N_b$$

The inference rule associated with this protocol step is:

$$\frac{A}{N_b}\ Fresh(N_b)$$

meaning that the intruder could get a fresh nonce (generated by B) from the protocol just by supplying it with an agent identity. The side condition $Fresh(N_b)$ stipulates the freshness[2] of the information N_b. Here is the scenario that illustrates such a situation:

$$\begin{array}{llcll} 1. & I(A) & \longrightarrow & B & : & A \\ 2. & B & \longrightarrow & I(A) & : & N_b \end{array}$$

This sequence stipulates that the intruder does a masquerade on the role A. First, the intruder sends to B the principal identifier A. According to the protocol, the principal B will react by generating a fresh nonce N_b and sending it on the network to A, hence to $I(A)$.

The third step in the Woo and Lam protocol of Table 1 is:

[1] Rules without side condition will be presented as follows: $\frac{p_1 \ldots p_n}{m}$

[2] The symbol $Fresh$ denotes a unary predicate that holds whenever its argument is fresh.

$$3.\ A \longrightarrow B : \{N_b\}_{k_{as}}$$

The inference rule associated with this protocol step is:

$$\frac{X}{\{X\}_{k_{as}}}$$

meaning that the intruder could supply the protocol with any information X and get it encrypted under the secret key shared between the server S and a principal A. At this level, we would like to pinpoint one important feature in our verification algorithm. The reader should notice that we used X in the inference rule instead of N_b. The rationale underlying this choice is that the principal A, at the third step of the protocol, replies with the message received at the second step (i.e. N_b) encrypted by k_{as}. Notice that N_b is a nonce generated and uttered by B and sent to A. Now, let us mention the following facts:

- The principal A has no preliminary knowledge on the effective value of the expected message at the second step of the protocol. Accordingly, A has no means to verify the value of the received message at that step.
- The freshness of N_b is a local property. In other words, the freshness of the nonce cannot be attested by any principal other than B. So, A cannot require the freshness of the message received at the second step of the protocol.
- For the sake of generality, we assume that the exchanged messages are not typed. Consequently, A is unable to verify the type of the message received at the second step of the protocol.

Owing to these three facts, A can verify neither the value, the freshness nor the type of the message received at the second step of the protocol. Consequently, A will accept any message at that step and will reply by sending the encrypted version of this message under the key k_{as}. This explains the rationale underlying the use of a variable message X instead of N_b. We will show further how fatal attacks could stem from this second protocol step.

Here is the scenario that exhibits how the intruder could instrument the protocol so as to get the message $\{X\}_{k_{as}}$ provided that he supplies the protocol with X:

$$\begin{array}{llcll} 1. & A & \longrightarrow & I(B) & : & A \\ 2. & I(B) & \longrightarrow & A & : & X \\ 3. & A & \longrightarrow & I(B) & : & \{X\}_{k_{as}} \end{array}$$

This sequence stipulates that the intruder will do a masquerade on the role B. First, the intruder will wait for A to initiate a protocol session i.e. to send the message A which will be intercepted by $I(B)$. Next, the intruder $I(B)$ will send a message X to the principal A. The latter will react innocently by sending the message $\{X\}_{k_{as}}$.

From now on, it is clear that the intruder has the ability to encrypt any known message X with the secret key k_{as} shared by the server and a principal A. Indeed, this ability follows from the three-steps sequence above. This highlights a weakness in the design of the Woo and Lam protocol. More dramatically, the intruder could encrypt any known message X by *any* shared key k_{ps} where p stands for any principal identifier. The reader should notice that in the protocol specification the principal identifier A is implicitly universally quantified over all principals.

Now, let us give some general comments regarding the inference rules and the associated scenarios. First, the agent identifiers as well as the introduced variables are implicitly universally quantified. Second, the generated scenarios constitute *proofs* of the *correction* of the inference rules. Actually, the scenarios are propagated during the deduction process.

The fourth step in the Woo and Lam protocol of Table 1 is:

$$4.\ B \longrightarrow S : \{A, \{N_b\}_{k_{as}}\}_{k_{bs}}$$

The inference rule associated with this protocol step is:

$$\frac{A \quad X}{\{A, X\}_{k_{bs}}}$$

meaning that the intruder could supply the protocol with a principal identifier, say A, and a known message X so as to get an encrypted version of the composition of these two messages under the key k_{bs}. Here is the scenario that illustrates such a situation:

$$\begin{array}{llcll} 1. & I(A) & \longrightarrow & B & : & A \\ 2. & B & \longrightarrow & I(A) & : & N_b \\ 3. & I(A) & \longrightarrow & B & : & X \\ 4. & B & \longrightarrow & I(S) & : & \{A, X\}_{k_{bs}} \end{array}$$

This sequence stipulates that the intruder will do a masquerade on the role A. First, the intruder sends to B the principal identifier A. According to the protocol, the principal B reacts by generating a fresh nonce N_b and sending it on the network to A, hence to $I(A)$. The next step is very sensible from the security standpoint. In fact, at the third step, the principal B is waiting for a message (N_b) encrypted under a key (k_{as})

that is locally unknown (from B standpoint). Nevertheless, the principal B has no means to check neither that the received message at the third step is effectively encrypted under k_{as} (k_{as} is unknown to B), nor that its content is N_b. Hence, the principal B can accept any message, say X, at this third step. Finally, the principal B reacts confidently with the message $\{A, X\}_{k_{bs}}$. This pinpoints a second weakness in the design of the Woo and Lam protocol.

The fifth step in the Woo and Lam protocol of Table 1 is:

$$5.\ S \longrightarrow B : \{N_b\}_{k_{bs}}$$

The inference rule associated with this protocol step is:

$$\frac{\{A, \{X\}_{k_{as}}\}_{k_{bs}}}{\{X\}_{k_{bs}}}$$

meaning that the intruder could supply the protocol with a message of the form $\{A, \{X\}_{k_{as}}\}_{k_{bs}}$ so as to get the message $\{X\}_{k_{bs}}$. Here is the scenario that illustrates such a situation:

$$\begin{array}{llcll} 4. & I(B) & \longrightarrow & S & : & \{A, \{X\}_{k_{as}}\}_{k_{bs}} \\ 5. & S & \longrightarrow & I(B) & : & \{X\}_{k_{bs}} \end{array}$$

This sequence stipulates that the intruder will do a masquerade on the role B. First, the intruder sends to the server the message $\{A, \{X\}_{k_{as}}\}_{k_{bs}}$. According to the protocol, the server decrypts such a message twice (using k_{bs} then k_{as}) and sends the sub-message X encrypted under the key k_{bs}. Notice that the server, at the fourth step, is normally expecting a message of the form $\{A, \{N_b\}_{k_{as}}\}_{k_{bs}}$. However, the server can check neither the value, the freshness nor the type of N_b. Accordingly, the server accepts any other information instead of N_b. This explains the rationale underlying the use of X instead of N_b in the previous inference rule.

From now on, it is clear that the intruder has extra decrypting abilities. In addition to the usual ones, the intruder can instrument the protocol to decrypt messages encrypted under keys that are unknown to him. For instance, assume that the intruder is wishing to decrypt a message of the form $\{X\}_{k_{as}}$. Recall also that the intruder may be a legal principal that shares a symmetric key k_{is} with the server. The intruder can merely supply the server with the message $\{A, \{X\}_{k_{as}}\}_{k_{is}}$ to get finally the message $\{X\}_{k_{is}}$ which he can decrypt to extract X. Here, the intruder has used the server as an oracle to decrypt messages. Notice that this example illustrates only one possible application (among many) of the previous inference rule. This pinpoints a third weakness in the design of the Woo and Lam protocol.

We sum up in Table 3 the whole proof system that has been generated from the Woo and Lam protocol of Table 1. The inference rules and the associated scenarios have been automatically generated from the protocol specification. The generation method will be explained further in the sequel.

3.3 Verification

In this section, we show how to combine roles and inference rules so as to perform the verification of a given cryptographic authentication protocol. To keep the presentation simple, we will confine ourselves to the analysis of authentication properties. We will show further how to easily extend our verification algorithm to deal with security properties.

A principal A wishing to prove his identity to a principal B, has to answer all the challenges uttered by B. Hence, the intruder can impersonate the principal A if he can answer all those challenges uttered by B to A. Thus, the verification principal consists in checking whether the intruder, with all its computation abilities, can answer such challenges.

Actually, the intruder computation abilities consist of:

- An initial knowledge generally made of the keys that the intruder shares with other principals, the server identity and other principal identities. Notice that this initial knowledge must be given explicitly within the protocol specification.
- The usual encryption, decryption, composition and decomposition abilities. Indeed, the intruder could encrypt and decrypt any message under known keys. In addition, he has the ability to compose (catenate) and decompose messages.
- Those computation abilities given by the protocol itself and captured by the deductive proof system.

Now we come to the explanation of the verification procedure. Starting from the protocol description, we extract the roles. Now, for each role[3], the intruder will attempt to answer all the challenges uttered so as to perform a masquerade. The intruder will progressively follow the role in a stepwise way. For a

[3]Actually, the verification involves only those roles that require identity proofs from other roles. In the case of the Woo and Lam protocol, only the role B is involved in the verification since the protocol deals with unidirectional authentication. In a bidirectional authentication protocol, the two principals are involved.

	Inference Rules	Scenarios
R_1	$\dfrac{}{A}$	1. $A \longrightarrow I(B) : A$
R_2	$\dfrac{A}{N_b}\ Fresh(N_b)$	1. $I(A) \longrightarrow B : A$ 2. $B \longrightarrow I(A) : N_b$
R_3	$\dfrac{X}{\{X\}_{k_{as}}}$	1. $A \longrightarrow I(B) : A$ 2. $I(B) \longrightarrow A : X$ 3. $A \longrightarrow I(B) : \{X\}_{k_{as}}$
R_4	$\dfrac{A, X}{\{A, X\}_{k_{bs}}}$	1. $I(A) \longrightarrow B : A$ 2. $B \longrightarrow I(A) : N_b$ 3. $I(A) \longrightarrow B : X$ 4. $B \longrightarrow I(S) : \{A, X\}_{k_{bs}}$
R_5	$\dfrac{\{A, \{X\}_{k_{as}}\}_{k_{bs}}}{\{X\}_{k_{bs}}}$	4. $I(B) \longrightarrow S : \{A, \{X\}_{k_{as}}\}_{k_{bs}}$ 5. $S \longrightarrow I(B) : \{X\}_{k_{bs}}$

Table 3: The Deductive Proof System Associated with the Woo and Lam Protocol

given step, if the role sends a message, the intruder will merely store it and hence will extend his knowledge. If the role is waiting for receiving some particular message, the intruder will attempt to generate it using his current computation abilities. In other words, the intruder will try to synthesize the required message using his current knowledge modulo some composition/decomposition and encryption/decryption steps, and also up to the use of the deductive proof system.

Now, let us see how this applies to the unidirectional Woo and Lam authentication protocol of Table 1. Notice that the only concerned role is B. The corresponding role is:

$$\begin{array}{lcl} Role(B) & = & \langle ?, A, A\rangle \\ & & \langle !, N_b, A\rangle \\ & & \langle ?, \{N_b\}_{k_{as}}, A\rangle \\ & & \langle !, \{A, \{N_b\}_{k_{as}}\}_{k_{bs}}, S\rangle \\ & & \langle ?, \{N_b\}_{k_{bs}}, S\rangle \end{array}$$

Actually, before starting to take up the various challenges uttered by B, we have first to proceed to what we call the *generalization* of the role B. This operation aims to replace some messages by message variables as done in the deductive proof system. The rationale underlying such substitutions is that these messages could not be verified from the content, structure, freshness and type standpoint. This operation will be formally defined in the next section. The role B is rewritten into:

$$\begin{array}{lcl} Role(B) & = & \langle ?, A, A\rangle \\ & & \langle !, N_b, A\rangle \\ & & \langle ?, X, A\rangle \\ & & \langle !, \{A, X\}_{k_{bs}}, S\rangle \\ & & \langle ?, \{N_b\}_{k_{bs}}, S\rangle \end{array}$$

Here, we replace the message $\{N_b\}_{k_{as}}$ by a message variable X since B does not know the key k_{as}, hence he could accept any other information.

First of all, let us initialize the knowledge of the intruder, noted KI, with the intruder initial knowledge. In other words, KI is set to $\{k_{is}, A, B, S\}$. This means that the intruder initially knows the identity of the server S, his shared key with S, the role A and the role B.

Now the verification amounts to a constraint satisfaction problem. In fact, we are going to collect constraints that reflect the requirements that makes the intruder take up successfully all the challenges uttered by a particular role.

In the first step of the role B, the principal is waiting for a message identifier, say A. To take up this challenge, the intruder has to generate such an identifier from KI modulo some composition/decomposition and encryption/decryption steps, and also up to the use of the deductive proof system. Such a constraint is written as:

$$(E_1) \qquad KI \models_R A$$

where R stands for the set of the rules $R_1, \ldots, R_5$ of Table 3 together with the usual composition/decomposition and encryption/decryption rules.

At the second step, B sends the nonce N_b over the network. Hence, the message becomes available to the intruder who extends his current knowledge to become $K_2 = KI \cup \{N_b\} = \{k_{is}, A, B, S, N_b\}$.

At the third step, the intruder must supply B with X. Notice that the current knowledge of the intruder at this step is K_2. Accordingly, this second constraint will be written as:

$$(E_2) \qquad KI \cup \{N_b\} \models_R X$$

At the fourth step, B offers the message $\{A, X\}_{k_{bs}}$ to the intruder. The latter updates his knowledge to become $KI \cup \{N_b, \{A, X\}_{k_{bs}}\}$.

Finally, the intruder has to supply the role B with $\{N_b\}_{k_{bs}}$ in order to achieve the masquerade. This last constraint will be written as:

$$(E_3) \qquad KI \cup \{N_b, \{A, X\}_{k_{bs}}\} \models_R \{N_b\}_{k_{bs}}$$

At this level, we are ready to check whether the intruder can or cannot impersonate the role A. This amounts to check the existence of solutions to the constraint set $\{E_1, E_2, E_3\}$ which is the following system:

$$\left\{ \begin{array}{lll} \{k_{is}, A, B, S, I\} & \models_R & A \\ \{k_{is}, A, B, I, S, N_b\} & \models_R & X \\ \{k_{is}, A, B, I, S, N_b, \{A, X\}_{k_{bs}}\} & \models_R & \{N_b\}_{k_{bs}} \end{array} \right\}$$

In fact, we use the resolution as a decision procedure for the simultaneous resolution of these constraints. The decision procedure yields a set of substitutions each of which denotes a potential flaw. Furthermore, the inference rules of the proof system are annotated by the associated scenarios. Thus, the application of these inference rules propagates these scenarios. Such a propagation yields also complete attack scenarios. To sum up, we extract automatically flaws and attack scenarios from protocol specifications.

In the case of the constraint set $\{E_1, E_2, E_3\}$, the algorithm produces many flaws and attack scenarios. Most of these attacks are new and completely different from those known up to now. In what follows, we present in Table 4 one among the many elegant attacks yielded by our algorithm.

The reader should notice that the attack reported in Table 4 is both elegant and sophisticated. The elegance of the attack stems from three facts:

1. First, the attack involves five sessions, hence we believe that the attack cannot be generated manually (at least in an easy way).

2. Second, notice that the intruder does not make any use of his key. Consequently, the intruder can be an illegal user of the system.

3. Third, the intruder impersonates A without opening any protocol session with the principal A. Hence, A could be not operational when the intruder performs the attack.

Here is the way the attack of Table 4 could be carried out: In the communication step "1.1.", the intruder begins a session with B in order to claim that his identity is A. In the step "1.2.", the intruder I intercepts the nonce N_b generated by B and send it to A. In the step "1.3.", I replies by a message *anything* which means any message since B cannot do any verification. In the step "1.4.", B must react according to the protocol by sending the message $\{A, anything\}_{k_{bs}}$. This message will be intercepted by the intruder. To finish this protocol run, the intruder ultimately needs to synthesize the message $\{N_b\}_{k_{bs}}$ in order to send it to B at the step "1.5.". To get this last required message, the intruder will open the sessions 2, 3 and 4. Notice that such a message could be get in a simpler way but for the sake of elegance we use all these sessions.

At the session 2, the intruder I waits for a principal C to initiate a session with another principal D. Then, he intercepts the message C and replies by the nonce N_b in order to get the message $\{N_b\}_{k_{cs}}$ at the step 3 of this session. This message will be used to generate the message $\{C, \{N_b\}_{k_{cs}}\}_{k_{cs}}$. Indeed, the intruder has to wait that the same principal C initiates another session with another principal E as shown in Table 4. At the session 4, the intruder wants to obtain the message $\{C, \{N_b\}_{k_{cs}}\}_{k_{bs}}$ which makes easy getting the message $\{N_b\}_{k_{bs}}$. For instance, the intruder replies by the message $\{C, \{N_b\}_{k_{cs}}\}_{k_{cs}}$ as the C's answer which is supposed to be $\{N'_b\}_{k_{cs}}$. At the session 5, the intruder gets easily the message $\{N_b\}_{k_{bs}}$ by sending the message obtained at the session 4 to the server S.

Finally, the intruder terminates the first session, step "1.5.", with B, by sending the message $\{N_b\}_{k_{bs}}$ as an S's response. Hence, from now on B is convinced that the initiator of the session one is A.

1.1	$I(A)$	$\longrightarrow$	B	:	A
1.2	B	$\longrightarrow$	$I(A)$	:	N_b
1.3	$I(A)$	$\longrightarrow$	B	:	$anything$
1.4	B	$\longrightarrow$	$I(S)$	:	$\{A, anything\}_{k_{bs}}$
2.1	C	$\longrightarrow$	$I(D)$	:	C
2.2	$I(D)$	$\longrightarrow$	C	:	N_b
2.3	C	$\longrightarrow$	$I(D)$	:	$\{N_b\}_{k_{cs}}$
3.1	C	$\longrightarrow$	$I(E)$	:	C
3.2	$I(E)$	$\longrightarrow$	C	:	$C, \{N_b\}_{k_{cs}}$
3.3	C	$\longrightarrow$	$I(E)$	:	$\{C, \{N_b\}_{k_{cs}}\}_{k_{cs}}$
4.1	$I(C)$	$\longrightarrow$	B	:	C
4.2	B	$\longrightarrow$	$I(C)$	:	N_b'
4.3	$I(C)$	$\longrightarrow$	B	:	$\{C, \{N_b\}_{k_{cs}}\}_{k_{cs}}$
4.4	B	$\longrightarrow$	S	:	$\{C, \{C, \{N_b\}_{k_{cs}}\}_{k_{cs}}\}_{k_{bs}}$
4.5	S	$\longrightarrow$	$I(B)$	:	$\{C, \{N_b\}_{k_{cs}}\}_{k_{bs}}$
5.4	$I(B)$	$\longrightarrow$	S	:	$\{C, \{N_b\}_{k_{cs}}\}_{k_{bs}}$
5.5	S	$\longrightarrow$	$I(B)$	:	$\{N_b\}_{k_{bs}}$
1.5	$I(S)$	$\longrightarrow$	B	:	$\{N_b\}_{k_{bs}}$

Table 4: An Attack of the Woo and Lam Authentication Protocol

4 Algorithm

The intent hereafter is to present the verification algorithm. First of all, we would like to present an algebraic formalization of the message domain. The latter is defined in terms of a many sorted ordered algebra.

Definition 4.1 (Signature) *An order-sorted signature Σ is a pair $((\mathcal{S}, \leq), \Sigma)$ where $(\mathcal{S}, \leq)$ is a partially ordered set of sorts (type names) and Σ is a set of function symbols. Each function symbol f is associated with an arity i.e. a finite word of the form $s_1 \times \ldots \times s_n \rightarrow s_{n+1}$ where s_i ranges over the domain $\mathcal{S}$. The function f is said to be of arity n. The term $s_1 \times \ldots \times s_n$ is called the domain of the function f while s_{n+1} is called the codomain of f. Functions with arity 0 are called constants.*

In order to lighten the notation, we will use, in the sequel, Σ to denote a signature $((\mathcal{S}, \leq), \Sigma)$.

Definition 4.2 (Σ-Algebra) *Let $((\mathcal{S}, \leq), \Sigma)$ be an order-sorted signature. An order-sorted Σ-algebra is a pair $(\mathcal{A},\mathcal{F})$ where $\mathcal{A}$ is an $\mathcal{S}$-indexed family of sets, say $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$, and $\mathcal{F}$ is a Σ-indexed set of functions $\{f_{\mathcal{A}} \mid f \in \Sigma\}$, such that for all sorts s and s', $s \leq s'$ implies $\mathcal{A}_s \subseteq \mathcal{A}'_s$. Moreover, if $f : s_1 \times \ldots \times s_n \rightarrow s_{n+1}$ then $f_{\mathcal{A}} : \mathcal{A}_{s_1} \times \ldots \times \mathcal{A}_{s_n} \rightarrow \mathcal{A}_{s_{n+1}}$.*

Definition 4.3 (Free Algebra) *Let $((\mathcal{S}, \leq), \Sigma)$ be an order-sorted signature and X be an $\mathcal{S}$-indexed set of variable identifiers or merely variables i.e. $X = \bigcup_{s \in \mathcal{S}} X_s$ such that $X \cap \Sigma = \emptyset$. The free Σ-algebra over X is the pair $(T_\Sigma(X), F_\Sigma)$ where:*

- *The $\mathcal{S}$-indexed family $T_\Sigma(X)$ is defined as the least set that satisfies:*
  - *For all $x \in X_s$, $x \in (T_\Sigma(X))_s$.*
  - *For all $cte :\rightarrow s \in \Sigma$, $cte \in (T_\Sigma(X))_s$.*
  - *For all $f : s_1 \times \ldots \times s_n \rightarrow s \in \Sigma$ and for all $(t_1, \ldots, t_n) \in ((T_\Sigma(X))_{s_1} \times \ldots \times (T_\Sigma(X))_{s_n})$, $f(t_1, \ldots, t_n) \in (T_\Sigma(X))_s$.*
- *The Σ-indexed set of functions F_Σ is made of functions of the form $f_{T_\Sigma(X)} : ((T_\Sigma(X))_{s_1} \times \ldots \times (T_\Sigma(X))_{s_n}) \rightarrow T_\Sigma(X)_s$ that take tuples of the form $(t_1, \ldots t_n)$ to $f(t_1, \ldots t_n)$.*

The next definition introduces formally the algebra of messages used within this work. First of all, we consider the following sorts: Msg, Key, $Nonce$, Agt, Nat and $Server$ that classify respectively messages, keys, nonces, agents, naturals and the server. We consider these function symbols: e, c, key,$nonce$, agt that correspond respectively to encryption, catenation, key generation, nonce generation and agent generation. In addition, we assume the existence of a constant s denoting the server. Let, 0 and $succ$ be the usual natural constructors. The arities of these functions are given below:

$$
\begin{array}{llll}
0 & : & & \rightarrow Nat \\
succ & : & Nat & \rightarrow Nat \\
agt & : & Nat & \rightarrow Agt \\
s & : & & \rightarrow Server \\
nonce & : & Nat & \rightarrow Nonce \\
e & : & Msg \times Key & \rightarrow Msg \\
c & : & Msg \times Msg & \rightarrow Msg \\
key & : & Agt \times Agt & \rightarrow Key
\end{array}
$$

Definition 4.4 (Message Algebra) *Let $\mathcal{S}_M$ be the sort set $\{Msg, Agt, Key, Nonce, Agt, Server, Nat\}$ and Σ_M be the function symbol set $\{e, c, key, nonce, agt, s, 0, succ\}$. Let X_M be an $\mathcal{S}_M$-indexed family of message variables. We endow $\mathcal{S}_M$ with a partial order $\leq_M$ which stipulates that $Agt \leq_M Msg$, $Key \leq_M Msg$, $Nonce \leq_M Msg$, $Server \leq_M Agt$ and closed under reflexivity. We define the algebra of messages $T_{\Sigma_M}(X_M)$ as the free Σ_M-algebra over X_M. The free terms of the latter algebra are called messages.*

In what follows, we present formal definitions of roles and protocols.

Definition 4.5 (Protocol) *A protocol is a sequence of communication steps. Each step is a triple of the form $\langle A, B, m\rangle$ where A and B are principal identifiers, and m is a message. A triple $\langle A, B, m\rangle$ has to be read as: A sends the message m to B.*

Definition 4.6 (Role) *A role is a sequence of communication steps. Each step is a triple of the form $\langle dir, m, P\rangle$ where P is a principal identifier, m is a message and $dir \in \{!, ?\}$. A triple $\langle ?, m, P\rangle$ has to be read as: wait for receiving the message m from the principal P. A triple $\langle !, m, P\rangle$ has to be read as: send the message m to the principal P.*

From the protocol definition, it is straightforward how we can extract roles. A role P, for instance, is extracted from the protocol by looking at all the steps (in the same order) in which the character P is used to point out the sender or the receiver.

The verification algorithm is reported in Tables 5 and 6. We assume that each principal comes with the

$$
\begin{array}{l}
Glan(R) = \\
\quad \textbf{case } R \textbf{ of} \\
\quad\quad \langle !, \_, \_\rangle R' \Rightarrow Glan(R') \\
\quad\quad \langle ?, m, \_\rangle R' \Rightarrow \{m\} \ \cup \ Glan(R') \\
\quad \textbf{else } \{\} \\
\quad \textbf{end}
\end{array}
$$

$$
\begin{array}{l}
Reduce(M) = \\
\quad \textbf{case } M \textbf{ of} \\
\quad\quad \{e(m,k), k\} \ \cup \ M' \Rightarrow Reduce(\{m,k\} \ \cup \ M') \\
\quad\quad \{c(m_1, m_2)\} \ \cup \ M' \Rightarrow Reduce(\{m_1, m_2\} \ \cup \ M') \\
\quad \textbf{else } M \\
\quad \textbf{end}
\end{array}
$$

$$
\begin{array}{l}
Flat(M) = \\
\quad \textbf{case } M \textbf{ of} \\
\quad\quad \{e(m,k)\} \ \cup \ M' \Rightarrow Flat(\{m,k\} \ \cup \ M') \\
\quad\quad \{c(m_1, m_2)\} \ \cup \ M' \Rightarrow Flat(\{m_1, m_2\} \ \cup \ M') \\
\quad \textbf{else } M \\
\quad \textbf{end}
\end{array}
$$

$$
\begin{array}{l}
Bind(M) = \\
\quad \textbf{let } X \ new \textbf{ in} \\
\quad\quad \textbf{case } M \textbf{ of} \\
\quad\quad\quad \{m\} \ \cup \ M' \Rightarrow \{(m, X)\} \ \cup \ Bind(M') \\
\quad\quad\quad \{\} \Rightarrow \{\} \\
\quad\quad \textbf{end} \\
\quad \textbf{end}
\end{array}
$$

$$
\begin{array}{l}
GenMessage(m, B) = \\
\quad \textbf{case } B \textbf{ of} \\
\quad\quad \{(m, X)\} \ \cup \ B' \Rightarrow X \\
\quad \textbf{else} \\
\quad\quad \textbf{case } m \textbf{ of} \\
\quad\quad\quad e(m1, k) \Rightarrow e(GenMessage(m_1, B), GenMessage(k, B)) \\
\quad\quad\quad c(m_1, m_2) \Rightarrow c(GenMessage(m_1, B), GenMessage(m_2, B)) \\
\quad\quad \textbf{else } m \\
\quad\quad \textbf{end} \\
\quad \textbf{end}
\end{array}
$$

Table 5: The Verification Algorithm: Part I

$GenRole(R, KI, Fresh) =$
  let $GenRoleStep(Role, K, F, Role1) =$
    let
      $M = Glan(Role1),$
      $RM = Reduce(M \cup K \cup F),$
      $B = Bind(RM \backslash (K \cup F)),$
      in case $Role$ **of**
        $\langle dir, m, A\rangle Role' \Rightarrow$
        $\langle dir, GenMessage(m, B), A\rangle GenRoleStep(Role', K, F, \langle dir, m, A\rangle Role1)$
      else $\Rightarrow \langle\rangle$
      end
    end
  in $GenRoleStep(R, K, F, \langle\rangle)$
  end

$RoleRules(GRole, KI, Fresh) =$
  let $RoleRulesStep(GR, K, F, Pr) =$
    case GR **of**
      $\langle ?, m, A\rangle R' \Rightarrow$
        $RoleRulesStep(R', K, F, Pr \cup \{m\})$
      $\langle !, m, A\rangle R' \Rightarrow$
        $\langle Pr, m', Flat(Pr \cup \{m'\}) \cap F\rangle \cup RoleRulesStep(R', K, F, Premises)$
      else $\Rightarrow \{\}$
    end
  in $RoleRulesStep(GRole, KI, Fresh, \{\})$
end

$Collect(Role, K, Rule)$
  case $Role$ **of**
    $\langle ?, m, \_\rangle Role' \quad \Rightarrow \quad \langle K, R, m\rangle Collect(Role', K, Rule)$
    $\langle !, m, \_\rangle Role' \quad \Rightarrow \quad Collect(Role', K \cup \{m\}, Rule)$
  else $\Rightarrow \langle\rangle$
  end

Table 6: The Verification Algorithm: Part II

specification of two sets $KI(R)$ and $Fresh(R)$. The set $KI(R)$ corresponds to the initial knowledge of the principal R. Generally, the set $KI(R)$ includes the identity of the role R, the server identity (if it exists), the identities of the other roles and all the keys that he shares with other principals. The set $Fresh(R)$ includes all those fresh messages generated by the role R.

In Tables 5 and 6, we present the main functions invoked by the verification algorithm. The functions reported in Table 5 operate on messages while the functions reported in Table 6 operate on roles.

First of all, let us explain the functions operating on messages. The Function *Glan*: takes as parameter a role, say R, and yields a message set, say M. The latter embodies all those messages received by R during all the protocol steps. Given a message set M, the Function *Reduce* computes a reduced message set by recursively decomposing and decrypting (with known keys) those messages in M. For instance, $Reduce(\{e(c(m, m_1), k_{as}), e(m_2, k_{ab}), k_{as}\})$ yields $\{m, m_1, e(m_2, k_{ab}), k_{as}\}$. The function *Reduce* is used essentially to perform role abstraction, therefore to generate rules of the inference system. Given a message set M, the function $Flat$ computes a flattened message set by recursively decomposing and decrypting non-elementary messages in M. For instance, $Flat(\{e(c(m, m_1), k_{as}), e(m_2, k_{ab}), k_{as}\})$ yields $\{m, m_1, m_2, k_{as}, k_{ab}\}$. The function $Flat$ is used to generate the side conditions of the inference rules. After the computation of the premise and the conclusion parts, the side condition is generated. For example, if we want to know the side condition of the rule $\frac{\{N_{a_1}, X\}_{k_{as}}}{X}$ and the set of fresh messages is $\{N_{a_1}, N_{a_2}, N_{a_3}\}$. Then, we compute $Flat(\{\{N_{a_1}, X\}_{k_{as}}, X\}) \cap \{N_{a_1}, N_{a_2}, N_{a_3}\} = \{N_{a_1}, X\} \cap \{N_{a_1}, N_{a_2}, N_{a_3}\} = \{N_{a_1}\}$ and we put as side condition $Fresh(N_{a_1})$. Hence, the rule the complete rule is $\frac{\{N_{a_1}, X\}_{k_{as}}}{X} Fresh(N_{a_1})$.

The Function *Bind* takes as parameter a message set M and yields a set of bindings. A binding is a pair (m, X) where the first component is a message and the second component is a message variable. The *Bind* function aims to bind all the messages in M to newly allocated message variables. Actually, in the verification algorithm, the *Bind* function will be invoked with the set of unknown messages to some role and hence replacing (or binding) these messages by (or to) variables. The Function *Gen* takes as parameter a message m together with a binding set B and yields the argument message m in which we replace all the messages occurring in B by their corresponding variables. Indeed, the function *Gen* rewrites a message according to a binding. For instance, given the binding $B = \{(e(m, k), X)\}$, the message $m = e(e(m, k), k')$ will be rewritten into $m' = e(X, k')$. The message m' denotes the interpretation of the message m by a role for which the key k is unknown.

Now, we would like to explain the functions that generate the rules and the constraint set. Recall that, the constraint set tells us which messages the intruder has to find in order to attack the protocol. The Function *GenRole* takes as parameters a role R, an initial knowledge (message set) KI and a set of fresh messages (those messages created by the protocol specifically for a particular session) $Fresh$. The computation is done step by step along the protocol steps. The function *GenRole* gives as outcome the generalized version of the role R. The computation of such a generalized role involves the computation of the corresponding binding and rewriting the role accordingly. Indeed, at first we collect M, the set of messages received by R before the current analyzed step. This is captured in the function *GenRoleStep* by *Role*1 which is initially $\langle\rangle$. Hence $M = Glan(Role1)$. To obtain the reduced set associated with M, say RM, and taking into account that we have an initial knowledge (message set) KI and a set of fresh messages $Fresh$, we compute $Reduced(M \cup KI \cup Fresh)$. At this point, it is straightforward to see that those messages in $RM \backslash (KI \cup Fresh))$ are unknown to the role R. Those messages are then used to compute the binding B according to which the transmitted message in the current step will be generalized. The inference rules are computed so as to associate one rule per protocol step. In other words, we associate one inference rule per output communication made by a particular role. The function $RoleRules(GRole, KI, Fresh)$ takes as input the generalized role *GRole*, the initial knowledge KI and the fresh informations $Fresh$ associated with one role. If in the current step the role sends a message, then the rule premises set includes all the generalized messages received by this role before this current step and the conclusion is the message sent at that step. The rule side condition is pointed out by looking if the premises set contains a fresh information vis-à-vis this role. However, if at the current step the role receives a message, we have only to update the premises set by adding this message. To generate the complete inference system, we have to apply the function $RoleRules(GRole, KI, Fresh)$ to all generalized roles in the protocol. For instance, we remark that we associate to each step in the protocol an inference rule, since whenever one role receives a message there is necessary another which has sent it. Furthermore, since there is one message sent per protocol step, we generate one rule for each protocol step.

The Function *Collect* is devoted to the collection of those constraints needed to perform a masquerade on a role R. The function *Collect* takes as parameter the generalized role R and the set of messages K, and returns the set of constraints. Initially, the message set K is set to $KI(R) \cup Fresh(R)$. Whenever the role R sends (output communication) a message, the intruder adds it to his knowledge K. When the role is waiting for some message, say m, the algorithm generates a constraint of the form $(K, Rules, m)$ which should be read as follows: the intruder has to synthesize the message m thanks to the proof system *Rules* (inference rules) and also by using his knowledge K.

5 Case Study

In this section, we illustrate the application of our algorithm to the corrected version of the Woo and Lam protocol of Table 2. We will see that new flaws have been automatically discovered. First of all, here is the roles as extracted by the algorithms:

$$Role(A) = \langle !, A, B\rangle\langle ?, N_b, B\rangle\langle !, \{N_b\}_{k_{as}}, B\rangle$$

$$\begin{array}{rcl} Role(B) & = & \langle ?, A, A\rangle \\ & & \langle !, N_b, A\rangle \\ & & \langle ?, \{N_b\}_{k_{as}}, A\rangle \\ & & \langle !, (A, B, \{N_b\}_{k_{as}}), S\rangle \\ & & \langle ?, \{A, N_b\}_{k_{bs}}, S\rangle \end{array}$$

$$Role(S) = \langle ?, (A, B, \{N_b\}_{k_{as}}), B\rangle\langle !, \{A, N_b\}_{k_{bs}}, B\rangle$$

Their associated generalized roles are respectively:

$$Role(A) = \langle !, A, B\rangle\langle ?, X, B\rangle\langle !, \{X\}_{k_{as}}, B\rangle$$

$$\begin{array}{rcl} Role(B) & = & \langle ?, A, A\rangle \\ & & \langle !, N_b, A\rangle \\ & & \langle ?, X, A\rangle \\ & & \langle !, (A, B, X), S\rangle \\ & & \langle ?, \{A, N_b\}_{k_{bs}}, S\rangle \end{array}$$

$$Role(S) = \langle ?, (A, B, \{X\}_{k_{as}}), B\rangle\langle !, \{A, X\}_{k_{bs}}, B\rangle$$

The generalized roles are used by the function $RoleRules(GRole, KI, Fresh)$ to generate the inference system. This step yields the rules shown in Table 7. Since the protocol is a one way authentication, the intruder try only to convince a principal playing a role B that it is a principal A. Let IS be the inference system and KI the intruder initial knowledge,according to the B's generalized role the constraint set is:

$$\left\{ \begin{array}{lll} KI & \models_{IS} & A \\ KI \cup \{N_b\} & \models_{IS} & X \\ KI \cup \{N_b\} & \models_{IS} & \{A, N_b\}_{k_{bs}} \end{array} \right.$$

The resolution step yields the two following attacks:

- **Attack 1:**

$$\begin{array}{llcll} 1.1 & I(A) & \longrightarrow & B & : A \\ 1.2 & B & \longrightarrow & I(A) & : N_b \\ 2.1 & B & \longrightarrow & I(C) & : B \\ 2.2 & I(C) & \longrightarrow & A & : A, N_b \\ 2.3 & B & \longrightarrow & I(C) & : \{A, N_b\}_{k_{bs}} \\ 1.3 & I(A) & \longrightarrow & B & : X \\ 1.4 & B & \longrightarrow & I(S) & : A, B, X \\ 1.5 & I(S) & \longrightarrow & B & : \{A, N_b\}_{k_{bs}} \end{array}$$

  In this attack, the intruder uses two sessions in order to convince a principal playing B's role that he is the principal A. Here, the intruder meets no difficulties to supply B with the required messages except the last one, $\{A, N_b\}_{k_{bs}}$. For that reason, the intruder waits for B to begin a session with some principal, say C. For instance, I intercepts the B's message and replies by A, N_a as a C nonce. Hence, it obtains the message $\{A, N_b\}_{k_{bs}}$ at step 2.3 by which it completes the masquerade in session one.

- **Attack 2:**

$$\begin{array}{llcll} 1.1 & I(A) & \longrightarrow & B & : A \\ 1.2 & B & \longrightarrow & I(A) & : N_b \\ 1.3 & I(A) & \longrightarrow & B & : X \\ 1.4 & B & \longrightarrow & S & : A, B, X \\ 2.1 & A & \longrightarrow & I(B) & : A \\ 2.2 & I(B) & \longrightarrow & A & : N_b \\ 2.3 & A & \longrightarrow & I(B) & : \{N_b\}_{k_{as}} \\ 3.1 & I(B) & \longrightarrow & S & : A, B, \{N_b\}_{k_{as}} \\ 3.2 & S & \longrightarrow & I(B) & : \{A, N_b\}_{k_{bs}} \\ 1.5 & I(S) & \longrightarrow & B & : \{A, N_b\}_{k_{bs}} \end{array}$$

  In this scenario, the main session is session 1 where I is trying to convince B that he is the agent A. The intruder I sends the identity of A to the principal B which replies by sending a nonce N_b. As explained previously, at step 1.3, I

	Inference Rules	Scenarios
R_1	$\frac{}{A}$	1. $A \longrightarrow I(B) : A$
R_2	$\frac{A}{N_b}\ Fresh(N_b)$	1. $I(A) \longrightarrow B : A$ 2. $B \longrightarrow I(A) : N_b$
R_3	$\frac{X}{\{X\}_{k_{as}}}$	1. $A \longrightarrow I(B) : A$ 2. $I(B) \longrightarrow A : X$ 3. $A \longrightarrow I(B) : \{X\}_{k_{as}}$
R_4	$\frac{A, X}{A, B, X}$	1. $I(A) \longrightarrow B : A$ 2. $B \longrightarrow I(A) : N_b$ 3. $I(A) \longrightarrow B : X$ 4. $B \longrightarrow I(S) : A, B, X$
R_5	$\frac{A, B, \{X\}_{k_{as}}}{\{A, X\}_{k_{bs}}}$	4. $I(B) \longrightarrow S : A, B, \{X\}_{k_{as}}$ 5. $S \longrightarrow I(B) : \{A, X\}_{k_{bs}}$

Table 7: The Deductive Proof System Associated with the Corrected Version of Woo and Lam Protocol

can send any arbitrary value X because B is unable to perform any verification on the received message. At step 1.4, B sends (A, B, X) to S. At this step, one can imagine that S refuses the received message, but this is not important for the rest of the attack scenario. After this step I needs $\{N_b\}_{k_{as}}$ in order to progress in the protocol's steps. That explains why a parallel session with A is initiated.

6 Conclusion

We reported in this paper a new algorithm for the formal automatic verification of authentication protocols. This algorithm does not necessitate any specification of any protocol property or invariant. It takes as parameter the protocol specification and generates the set of flaws, if any, as well as the corresponding attack scenarios. The algorithm involves three steps. First, protocol roles are extracted from the protocol specification. Second, the intruder abilities to perform communications and computations are generated from the protocol specification. In addition to the classical known intruder computational abilities such as encryption and decryption, we also consider those computations that result from different instrumentations of the protocol. The intruder abilities are modeled as a deductive proof system. Third, the extracted roles as well as the deductive system are combined to perform the verification. The latter consists in checking whether the intruder can answer all the challenges uttered by a particular role. If it is the case, an attack scenario is automatically constructed. To exemplify the usefulness and efficiency of our approach, we illustrated it on Woo and Lam authentication protocol. Abadi and Needham have shown that the protocol is insecure and they proposed a new corrected version. In this paper, we discovered new unknown flaws in the Woo and Lam protocol and even in the corrected version of Abadi and Needham. A prototype of this verification algorithm has been implemented in BNR-Prolog.

As future work, we intend to lift this verification algorithm to deal with key-exchange cryptographic protocols. Actually, the same principles apply to these protocols. The two first steps of our verification algorithm remain unchanged. The only step that need to be accommodated is the third verification step. Here, we still continue working with the deductive proof system and the intruder knowledge to capture all the computation abilities of the intruder.

References

[1] M. Abadi and Marc. R. Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the*

10th Annual ACM Symposium On Principles of Distributed Computing, pages 201–216, August 1991.

[2] Martin Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. Technical report, DEC/SRC, December 1996.

[3] Martin Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *the Proceedings of the Fourth ACM Conference on Computer and Communications Security*. ACM Press, April 1997.

[4] Martin Abadi and Roger Needham. Prudent Engineering Practice for Cryptographic Protocols. Technical report, SRC DIGITAL, June 1994.

[5] P. Bieber. A Logic of Communication in a Hostile Environment. In *Proceedings of the Computer Security Foundations Workshop III*, pages 14–22. IEEE Computer Society Press, 1990.

[6] Pierre Bieber and Nora Boulahia Cuppens. Formal Development of Authentication Protocols. In *Proceedings of the BCS-FACS 6th Refinemment Workshop on software Engineering and its Applications*, January 1994.

[7] Pierre Bieber, Nora Boulahia Cuppens, Thomas Lehman, and Erich van Wickeren. Abstract Machines for Communication Security. In *the Proceedings of the IEEE Computer Security Foundation Workshop VI*, June 1993.

[8] Dominique Bolignano. An Approach to the Formal Verification of Cryptographic Protocols. In *the Proceedings of the Third ACM Conference on Computer and Communications Security, CCS'96, New Delhi, India*, pages 106–118. ACM Press, 1996.

[9] Colin Boyd. Security Archituctures Using Formal Methods. *Journal on Selected Areas in Communications*, 11(5):694–701, June 1990.

[10] Colin Boyd. A Framework for Authentication. In *Proceedings of the European Symposuim on Research in Computer Security, ESORICS 92*, volume 648 of *Lecture Notes in Computer Science*, pages 273–292. Springer Verlag, November 1992.

[11] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. In *Proceedings of the Royal Society of London*, volume 426, pages 233–271, 1989.

[12] Michael Burrows, Martin Abadi, and Roger M. Needham. Rejoinder to Nessett. *ACM Operating Systems Review*, 24(2):39–40, April 1990.

[13] U. Carlsen. *Formal Specification and Analysis of Cryptographic Protocols.* PhD thesis, Thèse d'Informatique soutenue à l'Université PARIS XI, October 1994.

[14] John Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature. Unpublished article available at `http://dcpu1.cs.york.ac.uk/~jeremy`.

[15] K. Gaarder and E. Snekkenes. Applying a formal analysis technique to the CCITT X.509 Strong Two-Way Authentication Protocol. *Journal of cryptology, 3(2)*, pages 81–98, 1991.

[16] Paul Gardiner, Dave Jackson, Jason Hulance, and Bill Roscoe. Security Modelling in CSP and FDR: Deliverable Bundle 2. Technical report, Formal Systems (Europe) Ltd, April 1996.

[17] Paul Gardiner, Dave Jackson, and Bill Roscoe. Security Modelling in CSP and FDR: Deliverable Bundle 3. Technical report, Formal Systems (Europe) Ltd, July 1996.

[18] Janice I. Glasgow, Glenn H. MacEwen, and Prakash Panangaden. A Logic for Reasoning About Security. *ACM Transactions on Computer Systems*, 10(3):226–264, August 1992.

[19] V.D. Gligor and R. Kailar. On Belief Evolution in Authentication Protcols. In *Proceedings of the IEEE Computer Security Foundations Workshop IV, Franconia*, pages 103–116, June 1991.

[20] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, May 1990.

[21] C. A. R. Hoare. *Communating Sequential Processes.* Prentice Hall, 1985.

[22] R. A. Kemmerer. Using Formal Verification Techniques to Analyse Encryption Protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 134–139. IEEE Computer Society Press, 1987.

[23] R. A. Kemmerer. Analysing Encryption Protocols Using Formal Verification Techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.

[24] Armin Liebl. Authentication in Distributed Systems: A Bibliography. *Operating Systems Review*, 27(4):122–136, October 1993.

[25] Gavin Lowe. An Attack on the Needham-Schroeder Public Key Authentication Protocol. *Information Processing Letters*, 56(3):131–136, November 1995.

[26] Gavin Lowe. Breaking and Fixing the Needham Schroeder Public-Key Protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag, 1996.

[27] Gavin Lowe. Some new attacks upon security protocols. In *Proceedings of the Computer Security Foundations Workshop VIII*. IEEE Computer Society Pres, 1996.

[28] Gavin Lowe. SPLICE-AS: A Case Study in Using CSP to Detect Errors in Security Protocols. Technical report, Programming Research Group, Oxford, 1996.

[29] Wenbo Mao and Colin Boyd. Towards the Formal Analysis of Security Protocols. In *Proceedings of the Computer Security Foundations Workshop VI*, pages 147–158. IEEE Computer Society Press, 1993.

[30] Catherine Meadows. The NRL Protocol Analyser: An Overview. *Journal of Logic Programming*, 1994.

[31] Catherine Meadows. Formal Verification of Cryptographic Protocols: A Survey. In *Proceedings of Asiacrypt 96*, 1996.

[32] R. Milner. The Polyadic π-Calculus: A Tutorial. Technical report, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1991.

[33] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes. Technical report, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1989.

[34] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[35] Louise E. Moser. A Logic of Knowledge and Belief about Computer Security. In J. Thomas Haigh, editor, *Proceedings of the Computer Security Foundations Workshop III*, pages 57–63. IEEE, Computer Society Press of the IEEE, 1989.

[36] C. Meadows R. Kemmerer and Jonathan Millen. Three Systems for Cryptographic Protocol Analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[37] P. V. Rangan. An Axomiatic Basis of Trust in Distributed Systems. In *Proceedings of the 1988 Symposium on Security and Privacy*, pages 204–211. IEEE Computer Society Press, April 1988.

[38] Bill Roscoe and Paul Gardiner. Security Modelling in CSP and FDR: Final Report. Technical report, Formal Systems Europe, October 1995.

[39] A.D. Rubin and P. Honeyman. Formal Methods for the Analysis of Authentication Protocols. Technical Report Technical report 93–7, Technical Report, Center for Information Technology Integration, 1993. University of Michigan. Internal Draft.

[40] Steve Schneider. Security Properties and CSP. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 174–187. IEEE Computer Society Press, May 1996.

[41] Einar Snekkenes. Authentiction in Open Systems. *In 10th IFIP WG 6.1 Symposium on Protocol Specification, Testing and Verification*, pages 313–324, June 1990.

[42] Einar Snekkenes. *Formal Specification and Analysis of Cryptographic Protocols*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, Norwegian Defence Research Establishment, P.O. Box 25, N-2007, Kjeller, Norway, January 1995.

[43] Paul Syverson. The Use of Logic in the Analysis of Cryptographic Protocols. In Teresa F. Lunt and John McLean, editors, *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 156–170. IEEE Computer Society, May 1991.

[44] Paul Syverson. Knowledge, Belief, and Semantics in the Analysis of Cryptographic Protocols. *Journal of Computer Security*, 1(3):317–334, 92.

[45] Paul Syverson and Paul C. van Oorshot. On Unifying some Cryptographic Protocol Logics. In

Proceedings of the IEEE 1994 Computer Society Symposium on Security and Privacy, pages 14–28. IEEE Computer Society, May 1994.

[46] The commission of the European Communities CEC DG-XIII. Security Investigation Final Report. Technical Report S2011/7000/D010 7000 1000, CEC, September 1993.

[47] Vijay Varadharajan. Formal specification of a secure distributed system. *In Proceedings of the 12th National Computer Security Conference*, pages 146–171, October 1989.

[48] Vijay Varadharajan. Verification of network security protocols. *Computers and Security*, 8, December 1989.

[49] Vijay Varadharajan. Use of Formal Technique in the Specification of Authentication protocols. *Computer Standards and Interfaces*, 9:203–215, 1990.

[50] T. Y. C. Woo and S. S. Lam. Authentication for Distributed Systems. *Computer*, 25(1):39–52, January 1992.

[51] T. Y. C. Woo and S. S. Lam. A Lesson on Authentication Protocol Design. *Operating Systems Review*, pages 24–37, 1994.