The ISC Int'l Journal of Information Security

January 2016, Volume 8, Number 1 (pp. 3–24)

http://www.isecure-journal.org

A Short Introduction to Two Approaches in Formal Verification of Security Protocols: Model Checking and Theorem Proving

Mohsen Pourpouneh [1], and Rasoul Ramezanian [2,*]

[1] *Mathematical Sience Department, Sharif University of Thechnology, Tehran, Iran*

[2] *Mathematical Sience Department, Ferdowsi University of Mashhad, Mashhad, Iran*

ARTICLE INFO.

Article history:

Received: 4 September 2015

Revised: 10 December 2015

Accepted: 15 January 2016

Published Online: 17 January 2016

Keywords:

Cryptographic Protocols, Formal Verification, Model Checking, Theorem Proving.

ABSTRACT

In this paper, we shortly review two formal approaches in verification of security protocols; model checking and theorem proving. Model checking is based on studying the behavior of protocols via generating all different behaviors of a protocol and checking whether the desired goals are satisfied in all instances or not. We investigate *Scyther* operational semantics as an example of this approach and then we model and verify some famous security protocols using *Scyther*. Theorem proving is based on deriving the desired goals from assumption of protocols via a deduction system. We define a deduction system named *Simple Logic for Authentication* to formally define the notion of authenticated communication based on the structure of the messages, and then we investigate several famous protocols using our proposed deduction system and compare it with the verification results of *Scyther* model checking.

1 Introduction

Secure communication is an important issue since ancient times. The evergoing development of computer networks and software system, increases the likelihood of subtle errors in such systems. In order to provide secure communication among two or more agents, a lot of cryptographic protocols are designed and implemented. However, if a protocol is not designed correctly it may cause catastrophic loss of information. The adversary can replace or prevent some of the messages in a protocol and therefore prevent the protocol to reach its goals. Some of the most well known security flaws are false authentication and key compromised attacks. As an example, in 1978 Needham and Schroeder published a key distribution protocol [1] which is the basis for a whole class of related protocols. Although, in 1981 Denning and Sacco [2] proposed an attack over Needham and Schroeder protocol, which allowed the intruder to pretense an old, compromised key as a new key.

Due to the complexity of the environment in which the protocol is used, it is usually difficult to find the errors of a protocol manually or just by trial and error, or trusting the correctness of a protocol based on intuition and informal arguments. Accordingly, automatic methods are used for verifying the correctness of cryptographic protocols. One way of achieving to these methods is by using *formal methods* which are are mathematically based languages, techniques, and tools for describing, specifying and verifying cryptographic protocols.

Specifications is the process of formally define de-

* Corresponding author. The names are alphabetically sorted.
Email addresses: m_pourpouneh@mehr.sharif.edu (M. Pourpouneh), rramezanian@um.ac.ir (R. Ramezanian).

ISSN: 2008-2045

sired properties of a protocol (such as authentication, secrecy, anonymity, etc) via an appropriate language. Describing is the process of formally explaining the behavior of the protocol using mathematical objects. Verification is the process of proving that the protocol is correct and reaches the desired protocol goals.

The rest of this paper is organized as follow: In Section 2 we review different types of attacks as well as security properties of cryptographic protocols. Then in Section 3 we briefly discuss two different approaches which is used in formal verification of protocols. Section 4 reviews the *Scyther* model checker as one of the latest model checkers. In Section 5 we propose a deduction system named *Simple Logic for Authentication*, and in Section 6 we describe how to model and verify cryptographic protocols using *Simple Logic for Authentication*. Section 7 provides several different examples of how the protocols are verified in our proposed model. Finally, Section 8 concludes the paper.

2 Background

A *protocol* is a finite sequence of messages between two or more agents $A_1, A_2, \ldots, A_n$ as follows:

1. $A_{i_1} \mapsto A_{j_1} : m_1$
2. $A_{i_2} \mapsto A_{j_2} : m_2$
3. $A_{i_3} \mapsto A_{j_3} : m_3$
.. ...
k. $A_{i_k} \mapsto A_{j_k} : m_k$.

for some arbitrary $k \in \mathbb{N}$ where for every $1 \leq t \leq k$, $i_t, j_t \in \{1, 2, ..., n\}$. In order to analyze the behavior of a protocol, a protocol and its goals should be defined formally, therefore two things should be described at first 1- the assumptions of the protocol, and 2- the goals of the protocol.

As an example consider the Needham-Schroeder public key protocol [1]. Following, is the informal description of the protocol.

1. $A \mapsto B : \{A, N_A\}_{pk_B}$
2. $B \mapsto A : \{N_A, N_B\}_{pk_A}$
3. $A \mapsto B : \{N_B\}_{pk_B}$

Protocol 1. Needham-Schroeder public key.

The assumptions of this protocol is that the *nonces* generate by each agents are fresh, and every agent can verify their freshness. The other assumption is that, the *keys* are initially secret and they are unknown to the adversary. The goal of the protocol is mutual authentication, i.e., to assure each agent that they are communicating with the intended partner.

2.1 Security Attacks

In this section we review some list of typical weaknesses, that the protocols might be vulnerable against them. Based on the adversary activity the protocols can be divided in to two groups, *Passive* and *Active* attacks. In a passive attack the adversary does not interrupt the communications of the legitimate agents. But, in an active attack the adversary is online and he communicates with agents.

- **Eavesdropping** is a passive attack and it is the most basic attack that applies to every protocol. In this attack the adversary only eavesdrops the communication channel and sees the messages that are communicating between the agents. Generally, protocols are secured against this attack by using encryption.
- **Modification** in a protocol if the messages are not integrated or their fields are not redundant then the protocol might be vulnerable against modification attack. In modification attacks the adversary is not required to know the exact content of the messages. As an example, assume that the adversary flips some bits of a message which contains a session key. If the message is not integrated, then the message might yield another session key which is completely different from the original. It is shown that encryption is not enough for providing the required integrity. For more information of this the reader is referred to papers by Stubblebine and Gligor [3] and by Mao and Boyd [4].
- **Replay** the adversary records the messages and uses them at a later time in other protocols. The replay attack is an active attack. As an example consider the following protocol:

$A \mapsto B : \cdots$

$B \mapsto A : \{session\ Key\}_{k_{AB}}$

$A \mapsto B : \cdots$

Protocol 2. An example of a protocol vulnerable to replay attack.

Since, in this protocol A can not verify that the message is generated at the time of protocol run or prior to that, the adversary can replace this message with an old one, which causes the agents to have different key.

The solution to this problem is to use nonce or timestamps which shows that the messages are generated at the time of protocol run. In [5]

Syverson has produced a classification of different types of replay attacks.

- **Preplay** is an extension of replay attack. In this attack the adversary prepares for the attack by running a the same protocol or another protocol before running main protocol. An example of this attack is *triangle attack* of Burmester [6]
- **Reflection** is one of the most important special case of replay attack. This attack may be be possible only if parallel runs of the same protocol are allowed. As an example of this situation consider an Internet host, which accept sessions from multiple clients and uses the same identity and set of cryptographic keys for each run.

  As an example consider the following protocol from [7]:

$A \mapsto B : \{N_A\}_{k_{AB}}$

$B \mapsto A : \{N_B\}_{k_{AB}}, N_A$

$A \mapsto B : N_B$

Protocol 3. A protocol vulnerable to reflection attack.

When agent A receives the second message he concludes that the message is actually sent by B because he is the only one that knows k_{AB}. However, if two parallel run of this protocol is allowed then the adversary C can successfully complete two runs of the protocol as follow:

1. $A \mapsto C : \{N_A\}_{k_{AB}}$

1′. $C \mapsto A : \{N_A\}_{k_{AB}}$

2′. $A \mapsto C : \{N'_A\}_{k_{AB}}, N_A$

2. $C \mapsto A : \{N'_A\}_{k_{AB}}, N_A$

3. $A \mapsto C : N'_A$

3′. $C \mapsto A : N'_A$

Protocol 4. Reflection attack on Protocol 3.

In this case, A believes that he completed two run of the protocol with B whereas he is communicating with adversary and he has done all cryptographic tasks himself.

Reflection attack are also called 'oracle attacks', since of the agents acts as an oracle. A comprehensive treatment of reflection attacks is done by Bird *et al.* [8]

- **Denial of Service** (usually contracted to as *DoS attack*) are the attacks in which the adversary prevents the agents to complete the protocol. Denial of service usually occurs in practice against servers who are required to interact with many agents. However, it seems to be impossible to completely prevent denial of service attack, but there are certain methods to reduce the impact of denial of service attacks. Examples of these methods are Aura and Nikander [9], Meadows [10], and Juels and Brainard [11].
- **Typing Attacks** in practice when the agents receive a message they only see a string of bits and they have to interpret and separate the sub-messages. Typing attacks benefit from this fact and cause the agent to misinterpret a message and accept different sub-messages as each other.

  As an example, consider the Otway-Rees key transport protocol [12]:

1. $A \mapsto B : M, A, B, \{N_A, M, A, B\}_{k_{AS}}$
2. $B \mapsto S : M, A, B, \{N_A, M, A, B\}_{k_{AS}}, \{N_B, M, A, B\}_{k_{BS}}$
3. $S \mapsto B : M, \{N_A, k_{AB}\}_{k_{AS}}, \{N_B, k_{AB}\}_{k_{BS}}$
4. $B \mapsto A : M, \{N_A, k_{AB}\}_{k_{AS}}$

Protocol 5. Otway-Rees protocol.

A and B share long-term keys, K_{AS} and K_{BS} with server S, respectively. S generates a new session key k_{AB} and sends it to B. M and N_A are the nonces chosen by A, and N_B is the nonce chosen by B. Assuming that M is 64 bits, A and B 32 bits, and k_{AB} 128 bits, then the adversary can act in the following way:

1. $A \mapsto B : M, A, B, \{N_A, M, A, B\}_{k_{AS}}$
2. $B \mapsto S : M, A, B, \{N_A, M, A, B\}_{k_{AS}}, \{N_B, M, A, B\}_{k_{BS}}$
3. $S \mapsto B : M, \{N_A, k_{AB}\}_{k_{AS}}, \{N_B, k_{AB}\}_{k_{BS}}$
4. $C \mapsto A : M, \{N_A, M, A, B\}_{k_{AS}}$

Protocol 6. Typing attack on Otway-Rees protocol.

i.e., the adversary replaces the last part of the first message with the second part of the last messages, which causes A to believe that the new session key is the concatenation of M, A, B.

Type Confusion Attack type confusion attack, which is very similar to typing attack. The following example shows a protocol vulnerable to type confusion attack.

1. $B \mapsto A : m_1 = \{\uparrow_B, \downarrow_A, N_B\}_{K_{AB}}$
2. $A \mapsto B : m_2 = \{\uparrow_A, \downarrow_B, N_A, N_B\}_{K_{AB}}$
3. $B \mapsto A : m_3 = \{\uparrow_B, \downarrow_A, N_A + 1\}_{K_{AB}}$
4. $A \mapsto B : m_4 = \{\uparrow_A, \downarrow_B, K'_{AB}, N_B\}_{K_{AB}}$

Protocol 7. Protocol vulnerable to type confusion attack.

Assuming that nonce N_A and K'_{AB} has the same length, then the adversary can send message m_2 instead of m_4. The between type confusion attack and typing attack, is that it is possible to prevent typing attack by checking the components of a message, whereas in type confusion attack this should be considered as an implementation issue.

2.2 Security Goals

In this section we review some important security properties of protocols, which the designer of the protocol regards them as the goals of the protocol.

Secrecy expresses that certain information is not revealed to the adversary, even though this information is communicated over a network, which is under full control of adversary.

Authentication there exist many forms of authentication in literature. Informally, authentication is a simple statement about the existence of a communication partner. In every protocol, at least two agents are communicating. However, since the network is under complete control of the adversary, not every role execution guarantees that there actually has been a communication partner, and the message might be sent by the adversary. The following are four forms of authentication:

Aliveness is a form of authentication that aims to establish that an intended communication partner is 'alive'. In [13] defines four forms of aliveness, namely weak aliveness, weak aliveness in the correct role, recent aliveness and recent aliveness in the correct role.

Weak Aliveness this is the weakest form of authentication. If this property is satisfied in a protocol for a role, then it means that communication partner is alive.

Weak Aliveness in the Correct Role property express that the communication partner is alive and he is acting in his the role that is expected from the protocol description.

Recent Aliveness when weak aliveness or weak aliveness in correct role are satisfied, it means that the communication partner is alive and he has provided some messages which can be only generated by him. But, they do not tell any thing about 'when' these messages are generated. It is not known whether these messages are generated before, after, or during the protocol execution time. Recent aliveness expresses that the messages are generated at the time of the protocol execution.

Recent Aliveness in the Correct Role this security property is just a combination os the previous two property. Informally speaking, it means that the communication partner is alive, he is acting in the expected role and he is alive at the execution time of the protocol.

The relationship between aliveness properties are shown in Figure 1.

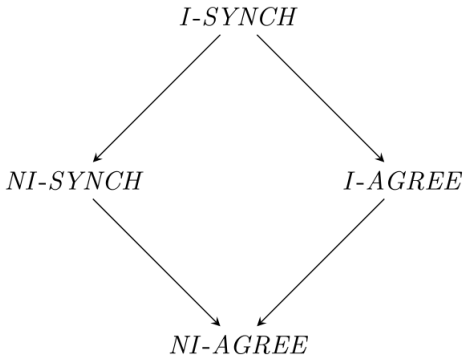

Figure 1. Aliveness authentication hierarchy.

The aliveness properties requires that some event to be executed by the communication partner, without putting restrictions on the contents of the exchanged messages. A much stronger authentication requirement is formed by *synchronisation*. Intuitively it requires that all received messages were indeed sent by the communication partner and that sent messages have indeed been received by the communication partner. This corresponds to the requirement that the actual message exchange has occurred exactly as specified by the protocol description [14].

Non-injective Synchronisation this property states that every thing that is intended to happen in the protocol description also happens in the protocol execution.

Injective Synchronisation since the agents may execute multiple runs of the same protocol or several protocols in parallel, possibly communicating with the same agents, the adversary may still be able to induce unexpected behaviour by replaying messages from

one session in another session. In particular, protocols satisfying non-injective synchronisation may still be vulnerable to replay attacks. This issue is overcomes by requiring that there is an injective mapping between the sent and received messages of each agent.

Message Agreement synchronisation ensures that the specified protocol behaviour occurs even in the presence of an adversary. The intuition behind agreement is that after execution of the protocol, the parties agree on the values of variables. The message agreement requires that the contents of the received messages correspond to the sent messages, as specified by the protocol. Therefore, it results in that the contents of every message in the protocol is exactly as specified by the protocol. Similar to synchronisation property it is possible to define Non-injective Agreement and Injective Agreement.

The relation among synchronisation and agreement properties are shown in Figure 2.

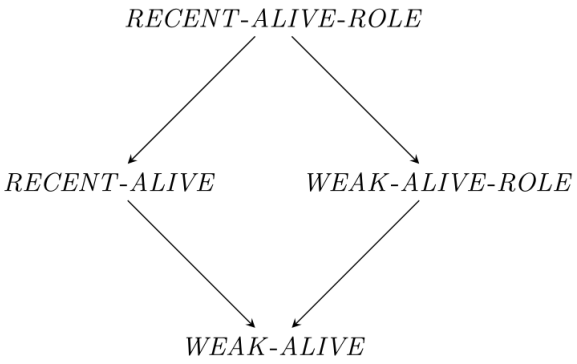

Figure 2. Synchronisation and agreement hierarchy.

3 Formal Verification Approaches

Generally speaking, in cryptographic protocol literature formal methods are mainly divided in to two categories:

3.1 Model checking

Model checking methods are techniques that build a relatively large, but finite, number of possible protocol behaviours. In general a model checker is a procedure that decides whether a given structure M is a model of a logical formula ϕ, in other words, whether M satisfies ϕ, abbreviated $M \models \phi$ [15]. Usually, in these methods M is an (abstract) model of the protocol, which is typically a finite automata-like structure, and ϕ is the desired property, which is typically expressed as a temporal or modal logic formula.

The protocol behavior is modeled as a directed graph in which every node represents different states of the protocol and edges are the possible transitions among different sates. The verification is performed with an exhaustive over the state space since the model is finite it is guaranteed that the search terminates, although it may take a long time even for simple protocols.

Model checking methods are generally fully automated and they are more suitable for finding attacks on protocols, however they are uninformative when the protocol is actually correct, An important limitation of the model checkers is that they only work for a number of finite parallel runs of a protocol, therefore when model checker does not report any attacks over a protocol it just means that the protocol is secure regarding a finite number of parallel run of the protocol or even when for multi-protocol verification i.e. when the protocol is run in parallel with other protocols.

The first model checkers based on temporal models are developed by Clarke and Emerson [16], Queille and Sifakis [17] and Pnueli [18] in 1980s. Some of the examples of model checkers are:

(1) **Murϕ** pronounced as Murphi is a general-purpose model checker. The Murphi description language is a high-level description language for finite-state asynchronous concurrent systems [19]. In 1997 Mitchell *et. al.* [20] used **Murϕ** in order to analyze Needham-Schroeder public key protocol [1], the TMN protocol [21] and the Kerberos protocol [22]. In each case **Murϕ** reported the previously known problems.

(2) **Brutus** is a model checking tool which is specially designed for cryptographic protocols analysis [23]. As well as application for key establishment protocols, Brutus is also used to analyze electronic payments protocols [23].

(3) **Scyther** is model checker developed in 2008 by Cas Cremers [13]. The novel features of Scyther is the possibility of unbounded verification of protocols with guaranteed termination, analysis of infinite sets of traces over the protocol graph, and support for multi-protocol environments [24]. In Section 4 we review Scyther in detail.

3.2 Theorem proving

These methods are generally more suitable for proving protocols correctness, rather than finding attacks on them. Theorem proving is usually based on formalisms such as first order logic or higher order logic. The

techniques that is used by these tools for proving the desired property are such as induction, rewriting, simplification and logical reasoning.

In theorem proving techniques the protocol and the required properties are modeled as logical propositions. Then a set of axioms and a set of inference rules are defined. Finally the property is correct (satisfied) if it can be inferred from the protocol by using defined axioms. Unlike to model checking methods, theorem proving techniques can be used for infinite state spaces. In contrast to model checkers, theorem proving methods are slow.

Some of the examples of model checkers are:

(1) **BAN Logic** in 1989, Burrows, Abadi and Needham (BAN) [25, 26] presented a logic for formal method for reasoning about authentication and key establishment protocols. BAN logic was the first formal verification method based on the beliefs of the agents participating in a protocol. The proofs constructed using BAN logic are usually short and even they can be obtained by hand. BAN logic assumes that authentication is a function of integrity and freshness, and uses logical rules to trace both of those attributes through the protocol [27].

BAN logic was successful to find flows is several protocols including Needham-Schroeder [1] and CCITT X.509 [28], and Needham-Schroeder, Kerberos [29], but it also has several known limitations including:

- BAN does not provide any way for reasoning about what agents does not know,
- BAN does not provide any way for converting a protocol to a idealized one,
- BAN is unable to represent whether an agent is honest or not.

Also, Nessett constructed a specific example BAN is unable to find the flaws which violate security in a basic sense [30]. In [31] Syverson explained a problem of informality in BAN logic's operational semantics and misunderstandings about BAN logic's goals. Due to this reasons Mao and Boyd [32] presented a set of measures to formalise BAN logic.

There are many extensions and improvements over BAN logic including the logics proposed by Gong, Needham, and Yahalom [33], Abadi and Tuttle [34], Syverson and van Oorschot [35], Kessler and Wedel [36], and van Oorschot [37].

(2) **Simple Logic for Authentication (SLA)** as a new theorem prover presented in this paper. The main idea behind *SLA* is to enable the agents to authenticate each message based on its contents and structure. In contrast to BAN logic and its extension, *SLA* is not based on the belief of agents, it is based on the structure of the messages. We review *SLA* in more detail in **??**.

4 Scyther Operational Semantics for Security Protocols

In Scyther [24] protocols are modeled explicitly by means of an operational semantics. The result is a role-based security protocol model that is agnostic with respect to the number of concurrent protocols. The security properties are modeled as local claim events. In this section we describe the operational semantic of *Scyther*.

4.1 Labelled transition system

In this section, we review operational semantics for security protocols which Scyther tool as a model checker works based on this semantics.

The operational semantics regards a protocol as a labeled transition system, and based on the trace space of the labelled transition system defines notions of securities.

A labelled transition system (LTS) is a concept for describing the behavior of a discrete system. A LTS consists of states and edges that connects states to each other. In an LTS the set of states, as well as the set of transitions might not be finite, or even countable. Formally, a LTS is a quadruple $(S, L, \rightarrow, s_0)$, in which

- S is the set of states;
- L is the set of labels;
- $\rightarrow$: $S \times L \times S$ is a ternary transition relation;
- $s_0 \in S$ is the initial state.

In this paper $(p, \alpha, q) \in \rightarrow$ is abbreviated as $p \xrightarrow{\alpha} q$ for $p, q \in S$ and $\alpha \in L$. A finite execution of a LTS $P = (S, L, \rightarrow, s_0)$, denoted by σ, is an alternating sequence of states and labels with start state s_0 and ending state s_n such that if $\sigma = [s_0, \alpha_1, s_1, \alpha_2, \ldots, \alpha_n, s_n]$ then $\forall i \leq i < n \ s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$. For the finite execution $[s_0, \alpha_1, s_1, \alpha_2, \ldots, \alpha_n, s_n]$ a finite trace of P is the sequence of the labels, i.e., $[\alpha_1, \alpha_2, \ldots, \alpha_n] \in L^*$.

A labelled transition system is defined by of a set of transition rules. A transition rule defines a number of premises $Q_1, Q_2, \ldots, Q_n (n \geq 0)$ which must all hold before a conclusion of the form $p \xrightarrow{\alpha} q$ is drawn:

$$\frac{Q_1, Q_2, \ldots, Q_n}{p \xrightarrow{\alpha} q}$$

Every security protocol describes a number of be-

haviours, called as roles. Every role in a protocol corresponds to a vertical axe of a Message Sequence Charts [1] . The system running the protocol may contain any number of agents. The same agent may execute any instances of roles. Each such instance is called a run. As an example, an agent can perform one initiator run and two responder run in parallel. The agents execute their runs to achieve some security goal. It is assumed that the adversary tries to oppose the agents security goals. Also, the agents taking part in a protocol run might be compromised by the adversary and try to invalidate the security goals of other agents. In order to prevent attacks, protocols use cryptographic primitives such as encryption or hash, signature. Throughout this paper we treat cryptographic primitives in a *black-box* approach. It is assumed that these primitives are idealized and we do not consider implementation errors.

In Scyther operational semantics, protocol specification describes the behaviour of every roles in the protocol. The specification of a protocol includes the initial knowledge of the roles, the declaration of functions, global constant and variables and the keys that are known to every role.

Agents execute the roles of the protocol. It is assumed that honest agents only follow the behavior described for their role in the protocol specification. For example in Needham-Schroeder public key 1, the agents are A and B, and they can have role *sender*/*receiver*. Note that each agent might be the sender in one run and the receiver in another parallel run.

The threat model used in this paper is the one suggested by Dolev and Yao in 1983 [39]. In this model the adversary is in full control of the communication network. The adversary is able to read, modify, create, and delete messages. Also, the adversary can compromise any number of agents and learn their secret keys.

The security requirements that are considered in this paper are safety properties, which ensures that nothing bad will happen.

4.2 Protocol Model

Every protocol is a set of messages terms exchange between the different roles in the protocol. The terms that are used in the specification of a protocol for different roles are called "role terms". The role terms is constructed using the following sets:

- *Var*, denotes variables that are used for storing the received messages,
- *Fresh*, denotes freshly generated values for each instantiation of a role,
- *Role*, denotes roles,
- *Func*, denotes function names.

Role terms are the basic term sets extended with constructors such as function application, term pairing, encryption.

$$\begin{aligned} RoleTerm ::= \; & Var|Fresh|Role|Func \\ & | \; Func(Roleterm) \\ & | \; (RoleTerm, RoleTerm) \\ & | \; \{|RoleTerm|\}_{RoleTerm} \end{aligned}$$

Encryption is defined as a function and the encrypted term can only be decrypted using the same term (for symmetric encryption) or the inverse key (for asymmetric encryption). Therefore, a function is defined to that inverses any role term: $\_^{-1} : RoleTerm \rightarrow RoleTerm$.

In *Scyther* the protocols are described as a set of roles, for each agent. Each role is able to execute a set of actions named *RoleEvent*. The set of events that can be executed by role R are as follows:

$$\begin{aligned} RoleEvent_R ::= \; & send_l(R, R', RoleTerm) \\ & | \; recv_l(R', R, RoleTerm) \\ & | \; claim_l(R, Claim[, RoleTerm]) \\ RoleEvent = \; & \bigcup_{R \in Role} RoleEvent_R \end{aligned}$$

in which event $send_l(R, R', RoleTerm)$ denotes the sending of message $RoleTerm$ by R, intended for R'. The event $recv_l(R, R', RoleTerm)$ the reception of message $RoleTerm$ by role R', apparently sent by role R. Event $claim_l(R, c, RoleTerm)$ expresses that R upon execution of this event expects security goal c to hold with optional parameter $RoleTerm$. A claim event denotes a local claim, which means that it only concerns role R and does not express any expectations at other roles. $l \in Lable$ are the labels which tags the events. Taging the events is necessary for two reasons: First, they are needed to disambiguate similar occurrences of the same event in a protocol specification. Second, they are used to express the relation between corresponding send and receive events.

As an example consider the following informal description of a protocol:

$$I \mapsto R : \left\{\{ni\}_{pk(R)}\right\}_{sk(I)}$$

This protocol contains two roles, i.e. an *initiator* and *responder*. ni is freshly generated nonce and $pk(\cdot)$ and

[1] Message Sequence Chart is an International Telecommunication Union standardised protocol specification language [38]

$sk(\cdot)$ are two functions. For this protocol we get the following protocol specification:

```
Role I {
   send_1(I, R, {{ni}_{pk(R)} }_{sk(I)});
}

Role R {
   reciv_1(R, I, {{V}_{pk(R)} }_{sk(I)});
claim(R, secret, V);
}
```

in which V is the variable that is used for storing the value of nonce ni.

A role specification is defined as a set of initial knowledge(the initial knowledge needed to execute the role), and a list of role events. A protocol specifies the behavior of a number of a number of roles by means of a partial function from the set *Role* to $RoleEvent^*$. Therefore, each role description corresponds to a (totally ordered) list of events.

A run is defined as an instantiated role. In order to instantiate a role, it should be binded to the names of actual agents and make the local constants unique for each instantiation.

In fact a role term is transformed into a run term by applying an *instantiation* from the set *Inst*, defined as

$$RID \times (Role \to Agent) \times (Var \to RunTerm)$$

The notation *runidof(inst)* denotes the run identifier from an instantiation *inst*.

4.3 Protocol Execution

In the previous section a formal notion for a protocol description is defined. The protocol description is a static description of how a protocol should behave. But, when a protocol description is executed, dynamic aspects are introduced. In this section we review how a protocol is executed in *Scyther*.

The semantics of a protocol description in the presence of an intruder is defined as the set of possible execution traces. A trace is a totally ordered set of events, in which no variables occur.

Each role in a protocol description can be executed any number of times, possibly in parallel, by an unbounded number of agents. The unique execution of a role in referred to as a *run*. A run is some specific instance of a role by an agent. Each run is identified using its identifier, which is a unique identifier at meta level, and it is used to distinguish between different runs. Each run can have unique variables and local uniquely generated values such as nonces. Executing a role turns a role description into a run. This process is referred to as instantiation. Roles can be instantiated multiple times in several runs. To ensure that the freshly generated values are unique for each instantiation, the identifier of the runs is appended to their names.

In order to define a protocol behaviour, the protocol descriptions (i.e., roles and their events) should be connected to its execution (i.e., instantiation of role events). Therefore, the combination of an instantiation function with a role event a is called *run event*.

As mentioned before, *Scyther* uses a labelled transition system of the form $(State, RunEvent, \to, s_0(P))$, for a given protocol P, in which:

The States are the set of possible states of the network of agents executing roles in a security protocol, $s_0(P)$ is the initial state of the protocol. The initial state of the system consists of the initial adversary knowledge and the empty set of runs. The initial adversary knowledge is the union of the fresh terms generated by the adversary, the set of agent names, and the initial knowledge of all compromised agents in all roles. The transition relation is defined in Figure 3. In each rule in this table, the left-hand side of the conclusion is state $\langle\langle AKN, F\rangle\rangle$, where AKN is the current intruder knowledge and F is the current set of active runs. The system transition from this state to the one on the right if the premises are satisfied. The new state contains an update of the set of active runs and the intruder knowledge. The transitions are labelled with the executed run events.

Match is a predicate used to match an incoming message to a pattern specified by a role term, in the context of a particular instantiation. Formally speaking, let $inst(\theta, \rho, \sigma), inst' = (\theta', \rho', \sigma') \in Inst$, $pt \in RoleTerm$ and $m \in RunTerm$, then the predicate $\text{Match}(inst, pt, m, inst')$ holds if and only if $\theta = \theta'$, $\rho = \rho'$ and

$$\begin{aligned}
&\langle inst' \rangle = m \wedge \\
&\forall v \in dom(\sigma') \ : \ \sigma'(v) \in type(v) \wedge \\
&\sigma \subseteq \sigma' \wedge \\
&dom(\sigma') = dom(\sigma) \sup vars(pt).
\end{aligned}$$

in which the *type* is a function which returns the type of a variable that can be *fresh*, *agent*, *Func*, *pk/sk* and symmetric key.

The $runsof(P, R)$ is the runs that can be created by a Protocol P for a role $R \in dom(P)$. Also, $runIDs(F)$ return the set of active run identifier for a given set of runs F.

$$[create_p]\frac{R \in dom(P) \quad ((\theta,\rho,\emptyset),s) \in runsof(P,R) \quad \theta \notin runIDs(F)}{\langle\langle AKN, F\rangle\rangle \xrightarrow{((\theta,\rho,\emptyset),\mathrm{create}(R))} \langle\langle AKN, F \cup \{((\theta,\rho,\emptyset),s)\}\rangle\rangle}$$

$$[send]\frac{e = send_l(R_1,R_2,m) \quad (inst,[e].s) \in F}{\langle\langle AKN, F\rangle\rangle \xrightarrow{(inst,e)} \langle\langle AKN \cup \{\langle inst\rangle(m)\}, (F\ \{(inst,[e].s)\}) \cup \{(inst,s)\}\rangle\rangle}$$

$$[recv]\frac{e = recv_l(R_1,R_2,pt) \quad (inst,[e].s) \in F \quad AKN \vdash m \quad \mathrm{Match}(inst,pt,m,inst')}{\langle\langle AKN, F\rangle\rangle \xrightarrow{(inst',e)} \langle\langle AKN, (F\ \{(inst,[e].s)\}) \cup \{(inst',s)\}\rangle\rangle}$$

$$[claim]\frac{e = claim_l(R,c) \ \vee\ e = claim_l(R,c,t) \quad (inst,[e].s)\ \in F}{\langle\langle AKN, F\rangle\rangle \xrightarrow{(inst,e)} \langle\langle AKN, (F\ \{(inst,[e].s)\}) \cup \{(inst,s)\}\rangle\rangle}$$

Figure 3. Transition rules of Scyther.

The $create_P$ rule demonstrates that in any state a new run from the set of possible runs $runsof(P,R)$ can be created. The only requirement for $create_P$ is that its run identifier θ should not already occur in F.

The premises of the *send* rule state that there is a run in current set of active runs (F), whose next step is a send event. Upon execution of this send event the adversary learns the sent message and the run progresses to the next step. The premises of the *recv* rule state that there is a run in F whose next step is a receive event. The difference with the send rule is that the transition is only enabled if the adversary can infer a message m that matches the pattern pt. If pt contains previously unbound variables, they are now instantiated by updating $inst$ to $inst'$. The adversary learns no new information and the run progresses. The premises of the *claim* rule state that there is a run in F whose next step is a claim event. Except for progressing the run, they have no effect on the state.

5 SLA: A Theorem Prover

Simple Logic for Authentication is a theorem prover which uses the structure of the messages to reason about them. In this section we first provide the intuition about an authenticated communication which forms the basis of *SLA*, and then we prove SLA semantics and finally give some examples of how SLA works.

5.1 The Intuition of Glass Pipe

We introduce a notion named glass pipe to explain our intuition of authentication, and based on this notion, we propose a simple logic to formally support our intuition of authentication.

Assume Alice and Bob want to share a key using the Diffie-Hellman key agreement protocol [40]; Alice wants to send the message g^a to Bob, and Bob wants to send the message g^b to Alice [2]. Alice and Bob seek for an *authenticated connection* to exchange their messages through it. They simply take a *glass pipe* with some balls satisfying the following three properties:

P1. ◦ Their glass pipe has just one tail, and just one head.
◦ It is only possible to input a ball to the glass pipe either through its head or its tail.
◦ The tail is only accessed by Bob and the head is only accessed by Alice.

P2. The radius of each ball is $\frac{2}{3}$ times of radius of the pipe, and thus at each time only one ball can pass through the pipe.

P3. The pipe is constructed by glass, and thus anybody (adversary) can see inside it, but it is not possible for anybody else (except Alice and Bob) to input balls to the pipe.

Alice generates a random number a, write g^a on a ball, and derives the ball away through the glass pipe to Bob. Bob receives the ball, generates a random number b, write g^b on another ball, and derives it away through the glass pipe to Alice. Then, they agree on g^{ab} as the session key.

The *glass pipe* with above three properties plays the role of an authenticated connection for Alice and Bob. Alice and Bob via the glass pipe, could complete a version of Diffie-Hellman key exchange protocol in an authenticated way. The authentication property in this version is concluded of three assumed properties P1, P2, and P3 for the glass pipe. We may say that the key exchange Diffie-Hellman in glass pipe has per-

[2] g is assumed as a generator of a multiplicative group Z_p^* for some large prime number p (where the discrete logarithm problem is hard), and a, and b are two random numbers generated by Alice and Bob, respectively.

fect forward secrecy [3] in the following meaning: If the glass pipe is broken at some time, as the construction of the glass pipe is independent of the agreed keys g^{ab}, and it is possible for the adversary to see inside the glass pipe, after breaking the pipe, the knowledge of the adversary does not change about the before agreed keys.

In the sequel, by the notion of the authenticated connection we mean a glass pipe with properties P1, P2, and P3.

Informally "Authentication" is the answer to the following question:

Who is on the other side of the *Glass Pipe*?

At the first look it may seem it is enough to answer to the question of "who constructed the message?" But, it is possible that the message is not freshly generated and it is sent by adversary. Knowing that the message is *freshly constructed* by the intended communication partner is not enough for authentication. There is a possibility that the actual "destination" for the message is some else, and the adversary redirected this message to the agent. Hence, the answer to the following questions provides the answer to " Who is on the other side of the line?"

(1) Who is the*source* of the received message?
(2) Who is the*destination* of the received message?
(3) Is the message constructed by the intended communication partner?
(4) Is the message *freshly* generated?
(5) Is the message protected against modification, i.e. is it *integrated*?

In the following sections we try to formally answer to these questions.

5.2 A Simple Logic

In this section, we propose our simple logic, **SL**, for formally resembling the notion of pipe glass via cryptographic attributes of messages. We first introduce the syntax of the logic, and then its deduction system.

5.2.1 Syntax

We first define a set of constant symbols, denoted by M, which formally presents the set of cryptographic messages.

We let $Agent = \{a_1, a_2, ..., b_1, b_2, ...c_1, c_2, ..\}$ be a set of symbols regarded as principle's name.

[3] A protocol is said to have perfect forward secrecy if compromise of long-term keys does not compromise past session keys

We let P be a set of symbols with regard to primitive messages containing following special kinds of symbols

- $ID_{pro}.ID_{ses}.ID_{step}$ which refers to the identification of a protocol, the identification of a session of the protocol, and the identification of the step of the protocol.
- For each $a \in Agent$, $\uparrow_a$ and $\downarrow_a$ are two symbols belong to P.

We let $Key = \{k^i_{ab}, sk^i_a, pk^i_a \mid i \in \mathbb{N}, a, b \in Agent\}$ be a set of symbols for keys where

- k^i_{ab} is a symbol to refer to the ith shared key between two agents a and b, assuming $k^i_{ab} = k^i_{ba}$.
- sk^i_a is a symbol for the ith secret-key of the agent a which corresponds to the symbol pk^i_a as the public-key.

We also assume $F = \{f_1, f_2, ...\}$ be a set of symbols regarded as one-way functions, and $HMAC = \{MAC_1, MAC_2, ...\}$ be a set of symbols regarded as hash message authenticated codes.

Definition 1. The set of constants symbols M of the **SL** logic, is defined as follows:

1- $P \cup Key \cup Agent \subseteq M$.
2- $m_1, m_2, ..., m_t \in M \Rightarrow tuple_t(m_1, m_2, ..., m_t) \in M$,
3- $m \in M \Rightarrow \{m\}_{k_{ab}}, \{m\}_{sk_a}, \{m\}_{pk_a} \in M$, for all $k_{ab}, sk_a, pk_a \in Key$,
4- $m \in M \Rightarrow MAC(k_{ab}, m), f(m) \in M$, for all $MAC \in HMAC$, $k_{ab}, sk_a \in Key$, and $f \in F$.
5- Each member of M is obtained inductively by items 1-4.

The following definition introduces the set of formulas of the **SL** logic.

Definition 2. The set of formulas, denoted by Formu, of the **SL** is defined as below.

$Hid(k)$, $Fresh_a(m)$, $Integ_a(m)$, $Const_b(a, m)$, $Sourc_b(a, m)$, $Dest_b(a, m)$, $Sub_a(d, m)$, $send_a(m)$, $Vaccess_a(b, d)$ and $PIPE(a, \langle d, m\rangle, b)$ are formulas in Formu where $k \in Key, a, b \in Agent$, and $d, m \in M$.

- $Hid(k)$ has to be read as '*the key k is hidden from adversary*'.
- $Fresh_a(m)$ has to be read as '*the principal a can verify that m is fresh*'.
- $Integ_a(m)$ informally means that '*the principal a can verify that m is integrated*'.
- $Const_b(a, m)$ informally means that '*the principal b can verify that whether principal a has the ability to construct m*'.
- $Dest_b(a, m)$ has to be read as '*the principal b can*

verify that the destination of the message m is the principal a'.

- $Source_b(a, m)$ has to be read as '*the principal b can verify that the source of the message m is the principal a*'.
- $Sub_a(d, m)$ informally means that '*the principal a can derive d as a sub-message of the message m*'.
- $send_a(m)$ means that '*principle a sends the message m through protocol.*
- $Vaccess_a(b, d)$ informally means that '*principal a can verify that only principals b has access to message d*
- $PIPE(a, \langle d, m\rangle, b)$ has to be read as '*the structure of the message m resembles a glass pipe such that the principal a sends the message d (where d is a sub-message of m) to the principal b through it*'.

The **SL** logic is a simple logic, where all formulas of **SL** are atomic [4]. Any formula of **SL** is simple; and contains no logical connectives as $\wedge, \vee, \neg, \rightarrow$, no quantifiers $\exists, \forall$, and no modal operators. It is why we call our logic *simple logic*.

5.2.2 Deduction System

In this part, we introduce the set of deduction rules of the **SL**. By these rules, we *formally* determine the meaning of the formulas of the **SL** logic.

- **Tuple rules:** Informally these rules enables the principles to be able to distinguish different parts of a messages, therefore it prevents attacks such as "typing attack".

  For every $t, j \in \mathbb{N}$, and for every $m, m_1, \ldots, m_t$, $d_1, d_2, \ldots, d_j \in M$,

  - $\dfrac{tuple_t(m_1, ..., m_t) = tuple_j(d_1, ..., d_j)}{t = j \quad m_1 = d_1 \ ... \ m_t = d_j}\ (p_1.rule)$
  - $\dfrac{}{tuple_1(m) = m}\ (p_2.rule)$
  - $\dfrac{m_1 = d_1 \ ... \ m_t = d_t}{tuple_t(m_1, ..., m_t) = tuple_t(d_1, ..., d_t)}\ (p_3.rule)$

  Sub-message rules: These rules allows the agents to verify and extract every single part of a recieved tuple.

  Let $m, m', d, d_1, d_2, ..., d_j \in M$ and $a, b \in Agent$, then for every $j \in \mathbb{N}$,

  - $\dfrac{}{Sub_a(d_i, tuple_j(d_1, d_2, \ldots, d_j))}\ (s_1.rule)$
  - $\dfrac{}{Sub_a(d, \{d\}_{k_{ab}})}\ (s_2.rule)$
  - $\dfrac{}{Sub_a(d, \{d\}_{sk_b})}\ (s_3.rule)$
  - $\dfrac{}{Sub_a(d, \{d\}_{pk_a})}\ (s_4.rule)$
  - $\dfrac{Sub_a(d, m) \quad Sub_a(m, m')}{Sub_a(d, m')}\ (s_5.rule)$

  Freshness rules: These rules ensures an agent that the received messages are fresh.
  For every $t \in \mathbb{N}$, for every $d, m_1, m_2, ..., m_t \in M$, for every $a, b \in Agent$, for every $k_{ab}, sk_b, pk_a \in Key$, and a known one-to-one function f (such as +, -, *,...),

  - $\dfrac{Fresh_a(d)}{Fresh_a(tuple_{t+1}(m_1, \ldots, d, \ldots, m_t))}\ (f_1.rule)$
  - $\dfrac{Fresh_a(d)}{Fresh_a(\{d\}_{sk_b})}\ (f_2.rule)$
  - $\dfrac{Fresh_a(d)}{Fresh_a(\{d\}_{pk_a})}\ (f_3.rule)$
  - $\dfrac{Fresh_a(d)}{Fresh_a(\{d\}_{k_{ab}})}\ (f_4.rule)$
  - $\dfrac{Fresh_a(d)}{Fresh_a(f(d))}\ (f_5.rule)$

  Construction rules: for every $d, m \in M$, for every $a, b \in Agent$, for every $MAC \in HMAC$, for every $k_{ab}, sk_a \in Key$,

  - $\dfrac{Hid(k_{ab})}{Const_b\big(a, (d, MAC(k_{ab}, d))\big)}\ (c_1.rule)$
  - $\dfrac{Hid(k_{ab})}{Const_b\big(b, (d, MAC(k_{ab}, d))\big)}\ (c_2.rule)$
  - $\dfrac{Hid(k_{ab})}{Const_b(a, \{d\}_{k_{ab}})}\ (c_3.rule)$
  - $\dfrac{Hid(k_{ab})}{Const_b(b, \{d\}_{k_{ab}})}\ (c_4.rule)$
  - $\dfrac{Hid(sk_a)}{Const_b(a, \{d\}_{sk_a})}\ (c_5.rule)$
  - $\dfrac{Fresh_a(d)\ Sub_a(d, \{m\}_{pk_a}) \quad Vaccess_a(b, d) \quad Hid(sk_a)}{Const_a(b, \{m\}_{pk_a})}\ (c_6.rule)$

  Informally rule c_6 means that if agent a can verify the freshness of message d and d is a one of the components of message m that is encrypted using his public key, and the only one who has access to d is agent b, then he is assured that $\{m\}_{pk_a}$ is

[4] Note that the language of **SL** does not have any predicate. For example, for every $k \in Key$, $Hid(k)$ is an atomic formula, and we do not have $Hid(-)$ as a predicate in **SL**.

For every $a, b \in Agent$, and $d, m \in M$,

$$\frac{Fresh_b(m) \quad Integ_b(m) \quad Sourc_b(a,m) \quad Sub_b(d,m) \quad Dest_b(b,m)}{PIPE(a, \langle d, m \rangle, b)} \ (pipe.rule) \tag{3}$$

constructed by agent b.

Integrity rules: for every $d \in M$, for every $a, b \in Agent$, for every $MAC \in HMAC$, for every $k_{ab}, sk_b \in Key$,

- $\dfrac{Hid(k_{ab})}{Integ_a(d, MAC(k_{ab}, d))} \ (i_1.rule)$
- $\dfrac{Hid(k_{ab})}{Integ_a(\{d\}_{k_{ab}})} \ (i_2.rule)$
- $\dfrac{Hid(sk_b)}{Integ_a(\{d\}_{sk_b})} \ (i_3.rule)$
- $\dfrac{Hid(k_{ab}) \quad Sub_a(d, d')}{Integ_a(d, \{d'\}_{k_{ab}})} \ (i_4.rule)$
- $\dfrac{Fresh_a(d) \quad Sub_a(d, \{m\}_{pk_a}) \quad Vaccess_a(b,d) \quad Hid(sk_a)}{Integ_a(b, \{m\}_{pk_a})} \ (i_5.rule)$

Informally rule i_5 means that if agent a can verify the freshness of message d and d is a one of the components of message m that is encrypted using his public key, and the only one who has access to d is agent b, then he is assured that $\{m\}_{pk_a}$ is integrated, and it has not been modified, since its modification requires to decrypt the message, but it is assumed that $sk(b)$ is hidden.

Destination rule: for every $a, b \in Agent$, for every $m \in M$,

- $\dfrac{Sub_b(\downarrow_a, m)}{Dest_b(a,m)} \ (d.rule)$

Source rule: for every $a, b \in Agent$, for every $m \in M$,

- $\dfrac{Sub_b(\uparrow_a, m) \quad Const_b(a,m)}{Sourc_b(a,m)} \ (so.rule)$

A common rule for Freshness and Integrity: for every $a \in Agent$, for every $d, m \in M$,

- $\dfrac{Fresh_a(m) \quad Sub_a(d,m) \quad Integ_a(m)}{Fresh_a(d)} \ (fi.rule)$

The $fi.rule$ means that if agent a can verify the integration and freshness of message m then every sub-message of m is also fresh.

Access rule: for every $a, b \in Agent$, for every $m \in M$,

- $\dfrac{Send_a(\{m\}_{pk_b}) \quad Sub_b(d,m) \quad Vaccess_a(a,d) \quad Hid(sk_b)}{Vaccess_a(b,d)} \ (ac.rule)$

The access rule shows how an agents shares a value with another agent.

Glass Pipe rule: The pipe rule is shown in Equation 3.

The glass pipe rule says that if d is sub-message of m, and principal b, from the structure of message m, can verify that

- m is fresh,
- m has integrity,
- the source of the message m is principal a,
- d is a sub-message of m, and
- the message m is sent for b (b is the destination of m),

then

the message m plays the function of a glass pipe that principal a has sent the message d to principal b through it.

The *glass pipe rule* is the core of our formal approach where we formally describe our intuition of how we can resemble a glass pipe.

- Formulas $Sourc_b(a,m)$, and $Dest_b(b,m)$ formalize property (P1) of glass pipe (see Section 5.1).
- $Fresh_b(m)$ formalizes property (P2),
- and formulas $Fresh_b(m)$, $Integ_b(m)$, $Sourc_b(a,m)$, and $Dest_b(b,m)$ all together formalize property (P3).

5.3 Complete Authentication

Glass pipe rule does not support security in the case of multi-session executions. For example, the protocol presented in Section 7.7 satisfies its goals, but there is an attack on in the situation of multi-session execution. To over come this condition, it is needed to add protocol ID, session ID and step ID. Therefore, we add a new predicate to the language of our proposed logic:

- $PSS_a(m)$ informally means that agent a can derive from structure of the message m that this message has sent according to which step of what protocol in what session.
- Protocol.Session.Step rule:

$$\frac{sub_a(ID_{pro}.ID_{ses}.ID_{step}, m)}{PSS_a(m)} \ (PSS.rule)$$

Therefore, the following rule, called *a.pipe rule*, becomes the main rule:

$$\frac{PSS_b(m) \quad PIPE(a, \langle d, m\rangle, b)}{Auth(a, \langle d, m\rangle, b)} \ (a.pipe\ rule)$$

6 How to Model and Verify a Security Protocol

In this section, we describe how to model and verify the security protocols.

A protocol PRO is assumed as a finite sequence of communications between two or more principals $A_1, A_2, ..., A_n$ as follows:

The symbol $M[PRO]$ refers to the set message $\{m_1, m_2, ..., m_k\}$ which are exchanged to complete the protocol. For each i, $1 \leq i \leq n$, $M_{A_i}[PRO] \subseteq M[PRO]$ is the set of all messages received by principal A_i.

To formally model a protocol two things should be described: 1- the assumptions of the protocol, and 2- the goals of the protocol.

1. **Assumptions of the protocol:**
   - **Nonce**. The freshness of Nonce which are generated by principals are considered as assumptions. For example, if n is a nonce generated by a principal a in a protocol, we consider $Fresh_a(n)$ as an assumption of the protocol.
   - **Keys**. keys which are initially assumed to be secret from the adversary in a protocol are assumed to be hidden. For example, if sk_a is a secret key of the principal a in a protocol, we regard $Hid(sk_a)$ as an assumption of the protocol.
2. **Goals of the protocols:** the goals of the protocol are the set of those messages which the protocol is designed to exchange them between some principals in an authenticated manner (for example $PIPE(A_{i_t}, \langle d, m_t\rangle, A_{j_t})$ is used to assert that the message d is transferred through a glass pipe from principal A_{i_t} to principal A_{j_t} where the message m_t plays the function of the glass pipe.

We refer to the set of all assumption of a protocol PRO by Γ_{pro} where the elements of Γ_{pro} are all atomic formulas like $Fresh_a(d)$ and $Hid(k)$ for $a \in Agent$, $d \in M$, and $k \in Key$. We also refer to goals of the protocol by G_1, G_2, ..., G_l, where each G_i is a formula like $PIPE(...)$.

In this way, a *formal model* of a protocol is a tuple

$$\langle \Gamma_{pro}, G_1, G_2, ..., G_l\rangle$$

where Γ_{pro} is the set of *assumptions*, and each G_i is a *goal* of the protocol.

Verifying a protocol model $\langle \Gamma_{pro}, G_1, G_2, ..., G_l\rangle$ is to show that each goal G_i $(1 \leq i \leq l)$ can be derived from the set of assumptions Γ_{pro} using the deduction system of our simple logic **SL**.

6.1 Some Examples of Modeling and Verifying

To illustrate the operation of modeling and verifying protocols via **SL**, in following, we formally model and verify two simple protocols. The first protocol is a key exchange protocol and the second protocol is the Needham-Schroeder public key protocol.

6.1.1 A Key Exchange Protocol

The following protocol is an authenticated version of Diffie-Hellman key exchange protocol.

- **Informal Description of the Protocol**:

1. $A \mapsto B : m_1$ (where $m_1 = \{tuple_4(\uparrow_A, \downarrow_B, n_i, g^i)\}_{sk_A}$)
2. $B \mapsto A : m_2$ (where $m_2 = \{tuple_5(\uparrow_B, \downarrow_A, n_i, n_r, g^r)\}_{sk_B}$)
3. $A \mapsto B : m_3$ (where $m_3 = \{tuple_6(\uparrow_A, \downarrow_B, n_r, n_i, g^r, g^i)\}_{sk_A}$)

Protocol 8. Diffie-Hellman key exchange protocol.

- **Formal Model of the Protocol:** the set of assumptions of the above protocol is

  $\Gamma = \{Fresh_A(n_i), Fresh_B(n_r), Hid(sk_A), Hid(sk_B)\}$.

  We assume $f_x(y) = x^y$ as symbols for one-way function. Goals of the protocol are

  $$G_1 = PIPE(A, \langle f_g(i), m_1\rangle, B),$$

  and

  $$G2 = PIPE(B, f_g(r), m_2, A)$$

- **Verification:** Applying the deduction rules of our proposed logic, we prove $\Gamma \vdash G1$, and $\Gamma \vdash G2$.

C1. $\Gamma \vdash Fresh_B(m_3)$ (by rules f1, f2 and assumption $Fresh_B(n_r)$).
C2. $\Gamma \vdash Integ_B(m_3)$ (by rule i3 and assumption $Hid(sk_A)$).
C3. $\Gamma \vdash Fresh_B(n_i)$ (by rule fi and consequences C1, and C2).
C4. $\Gamma \vdash Fresh_B(m_1)$ (by rules f1,f2 and consequence C3).
C5. $\Gamma \vdash Integ_B(m_1)$ (by rule i3 and assumption $Hid(sk_A)$).
C6. $\Gamma \vdash Const_B(A, m_1)$ (by rule c5 and assumption $Hid(sk_A)$).
C7. $\Gamma \vdash Sub_B(f_g(i), m_1)$ (by rules s1, s3, s5).
C8. $\Gamma \vdash Dest_B(B, m_1)$ (by rule d).

C9. $\Gamma \vdash PIPE(A, \langle f_g(i), m_1\rangle, B)$ (by pipe.rule).

6.1.2 Needham-Schroeder Protocol

The Needham-Schroeder public key protocol [1] is intended to provide mutual authentication between two parties. The protocol was proved to be correct via BAN logic but Lowe found that the protocol is vulnerable to a man in the middle attack [41]. We show that it is not verified in **SL**, either.

- **Informal Description of the Protocol**:

1. $A \mapsto B : m_1 = \{\uparrow_A, N_A\}_{pk_B}$
2. $B \mapsto A : m_2 = \{N_A, N_B\}_{pk_A}$
3. $A \mapsto B : m_3 = \{N_B\}_{pk_B}$

Protocol 9. Needham-Schroeder public key protocol.

- **Formal Model of the Protocol:** the set of assumptions of the Needham-Schroeder protocol is

$$\Gamma_{NS} = \{Hid(sk_A), Hid(sk_B), Fresh_A(N_A), Fresh_B(N_B)\}.$$

Goals of the protocol are

$$G_1 = PIPE(A, \langle N_B, m_3\rangle, B),$$

and

$$G2 = PIPE(B, \langle N_A, m_2\rangle, A).$$

- **Verification:** We show that $\Gamma_{NS} \nvdash G1$, and $\Gamma_{NS} \nvdash G2$. It is obvious that $\Gamma_{NS} \nvdash G2$ since I can not verify $Dest_A(A, \{N_A, N_B\}_{pk_A})$ and $source_A(B, \{N_A, N_B\}_{pk_A})$
- **Modification:** We modify the structure of messages of the Needham-Schroeder protocol such that goals G_1 and G_2 can be derived from Γ_{NS}.

1. $A \mapsto B : m_1 = \{\uparrow_A, \downarrow_B, N_A\}_{pk_B}$
2. $B \mapsto A : m_2 = \{\uparrow_B, \downarrow_A, N_A, N_B\}_{pk_A}$
3. $A \mapsto B : m_3 = \{\uparrow_A, \downarrow_B, N_B\}_{pk_B}$

Protocol 10. Modified Needham-Schroeder public key protocol.

Now we can verify that

$$G_1 = PIPE(A, \langle N_B, m_3\rangle, B)$$

C1. $\Gamma \vdash Fresh_B(m_3)$ (by rules f1, f3 and assumption $Fresh_B(N_B)$).
C2. $\Gamma \vdash Sub_B(N_B, m_3)$ (by rules s1, s4).
C3. $\Gamma \vdash Dest_B(B, m_3)$ (by rule s1, s4 and d).
C4. $\Gamma \vdash Vaccess_B(B, N_B)$ (by assumption $Hid(sk_A)$ and ac rule).
C5. $\Gamma \vdash Const_B(A, m_3)$ (by C1, C4, c6 and assumption $Fresh_B(N_B)$).
C6. $\Gamma \vdash Source_B(A, m_3)$ (by rules so and C5).
C7. $\Gamma \vdash Integ_B(m_3)$ (by rule i5 and C5).
C8. $\Gamma \vdash PIPE(A, \langle N_B, m_3\rangle, B)$ (by pipe.rule).

- **Resistance against known attacks:** Lowe found that this protocol is vulnerable to a man in the middle attack since the sender in the second message is not determined. In our modification in order to achieve PIPE property, we need to specify sender and receiver of a each message. So, this attack does not apply to our modified protocol.

7 Modeling and Verification of Some Protocols

In this section, in order to explain both theorem prover and model checking approaches, we model and verify some security protocols.

7.1 Bellare-Rogaway MAP1 Protocol

The MAP1 is a mutual authentication protocol presented by Bellare and Rogaway [42].

- **Informal Description of the Protocol**:

1. $A \mapsto B : m_1 = N_A$
2. $B \mapsto A : m_2 = N_B, \{\uparrow_B, \downarrow_A, N_A, N_B\}_{K_{AB}}$
3. $A \mapsto B : m_3 = \{\uparrow_A, N_B\}_{K_{AB}}$

Protocol 11. MAP1 protocol.

- **Formal Model of the Protocol:** the set of assumptions of the Bellare-Rogaway MAP1 protocol is

$$\Gamma_{BR} = \{Hid(K_{AB}), Fresh_A(N_A), Fresh_B(N_B)\}.$$

Goals of the protocol are

$$G_1 = PIPE(B, \langle N_A, m_2\rangle, A),$$

and

$$G_2 = PIPE(A, \langle N_B, m_3\rangle, B).$$

Informally speaking, goal G_1 is modeling the case in which agent A gives his nonce to B and in order to verify him, asks B to send this message to him trough a *glass pipe*.

- **Verification:** We show that $\Gamma_{BR} \vdash G_1$, but $\Gamma_{BR} \nvdash G_2$.

C1. $\Gamma \vdash Fresh_A(m_2)$ (by rules f1 f4 and assumption $Fresh_A(N_A)$).
C2. $\Gamma \vdash Integ_A(m_2)$ (by rule i2 and assumption $Hid(K_{AB})$).

C3. $\Gamma \vdash Const_A(B, m_2)$ (by rule c3 and assumption $Hid(K_{AB})$).
C4. $\Gamma \vdash Sub_A(\uparrow_B, m_2)$ (by rules s1, s2, s5).
C5. $\Gamma \vdash Source_A(B, m_2)$ (by rules so, C3 and C4).
C6. $\Gamma \vdash Sub_A(N_A, m_2)$ (by rules s1, s2, s5).
C7. $\Gamma \vdash Dest_A(A, m_2)$ (by rule d).
C8. $\Gamma \vdash PIPE(B, \langle N_A, m_2\rangle, A)$ (by pipe.rule).

In order to see that $\Gamma_{BR} \overset{?}{\vdash} G_2$ we should have $\Gamma \vdash Dest_B(B, m_3)$ but since B cannot verify $Sub_B(\downarrow_B, m_3)$ we have $\Gamma_{BR} \nvdash G_2$.

- **Modification:** We modify the structure of messages of Bellare-Rogaway MAP1 protocol such that goals G_1 and G_2 can be derived from Γ_{BR}.

1. $A \mapsto B : m_1 = tuple_1(N_A)$
2. $B \mapsto A : m_2 = tuple_2(tuple_1(N_B), \{tuple_4(\uparrow_B, \downarrow_A, N_A, N_B)\}_{K_{AB}})$
3. $A \mapsto B : m_3 = \{tuple_3(\uparrow_A, \downarrow_B, N_B)\}_{K_{AB}}$

Protocol 12. Modified MAP1 protocol.

It is obvious that we have $\Gamma_{BR} \vdash G_1$ and $\Gamma_{BR} \vdash G_2$.

- **Resistance against known attacks:** Alves-Foss [43] show that Bellare-Rogaway MAP1 is vulnerable against a chosen protocol attack. This attack does not apply to our modified version.
- **Code for verification using Scyther**

```
usertype SessionKey;
#const Fresh: Function;

protocol MAP1(A,B)
{
   role A
   {
      fresh NA: Nonce;
      var NB: Nonce;

      send_1(A,B, NA );
      recv_2(B,A, NB,{B,A,NA,NB}k(A,B) );
      send_3(A,B, {A,B,NB}k(A,B) );
      claim_A1(A,Nisynch);
}

   role B
   {
      var NA: Nonce;
      fresh NB: Nonce;

      recv_1(A,B, NA );
      send_2(B,A, NB,{B,A,NA,NB}k(A,B) );
      recv_3(A,B, {A,B,NB}k(A,B) );
      claim_B1(B,Nisynch);
   }
}
```

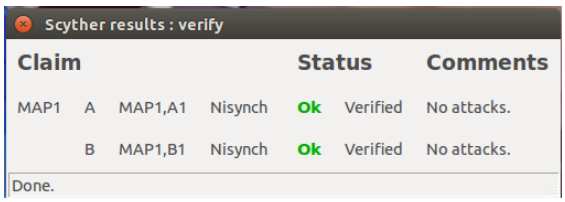

Figure 4. Simulation results for Bellare-Rogaway MAP1 protocol.

- **Result of the simulation**

7.2 Andrew Secure RPC Protocol

The Andrew Secure RPC protocol performs handshake using a key that both parties already share and then establishes a new new session key [44].

- **Informal Description of the Protocol**:

1. $A \mapsto B : m_1 = \{N_A\}_{K_{AB}}$
2. $B \mapsto A : m_2 = \{N_A + 1, N_B\}_{K_{AB}}$
3. $A \mapsto B : m_3 = \{N_B + 1\}_{K_{AB}}$
4. $B \mapsto A : m_4 = \{K'_{AB}, N'_B\}_{K_{AB}}$

Protocol 13. Andrew secure RPC protocol.

- **Formal Model of the Protocol:** the set of assumptions of the Andrew Secure RPC protocol is

$$\Gamma_{Andrew} = \{Hid(K_{AB}), Fresh_A(N_A), Fresh_B(N_B), Fresh_B(N'_B)\}.$$

Goals of the protocol are

$$G_1 = PIPE(A, \langle N_B + 1, m_3\rangle, B),$$
$$G_2 = PIPE(B, \langle N_A + 1, m_2\rangle, A),$$

and

$$G_3 = PIPE(B, \langle K'_{AB}, m_4\rangle, A).$$

- **Verification:** It is easily seen that $\Gamma_{Andrew} \nvdash G_1$, $\Gamma_{Andrew} \nvdash G_2$, and $\Gamma_{Andrew} \nvdash G_3$.
- **Modification:** We modify the structure of messages of Andrew Secure RPC protocol such that goals G_1, G_2, G_3 can be derived from Γ_{Andrew}.

1. $A \mapsto B : m_1 = \{tuple_3(\uparrow_A, \downarrow_B, N_A)\}_{K_{AB}}$
2. $B \mapsto A : m_2 = \{tuple_4(\uparrow_B, \downarrow_A, N_A + 1, N_B)\}_{K_{AB}}$
3. $A \mapsto B : m_3 = \{tuple_3(\uparrow_A, \downarrow_B, N_B + 1)\}_{K_{AB}}$
4. $B \mapsto A : m_4 = \{tuple_5(\uparrow_B, \downarrow_A, K'_{AB}, N'_B, N_A)\}_{K_{AB}}$

Protocol 14. Modified Andrew secure RPC protocol.

Now we prove that $\Gamma_{Andrew} \vdash G_1$, $\Gamma_{Andrew} \vdash G_2$, and $\Gamma_{Andrew} \vdash G_3$.

C1. $\Gamma \vdash Fresh_B(m_3)$ (by rules f1, f4, f5 and assumption $Fresh_B(N_B)$).
C2. $\Gamma \vdash Integ_B(m_3)$ (by rule i2 and assumption $Hid(K_{AB})$).
C3. $\Gamma \vdash Const_B(A, m_3)$ (by rule c3 and assumption $Hid(K_{AB})$).
C4. $\Gamma \vdash Sub_B(\uparrow_A, m_3)$ (by rules s1, s2, s5).
C5. $\Gamma \vdash Source_B(A, m_3)$ (by rules so, C3 and C4).
C6. $\Gamma \vdash Sub_B(N_B, m_3)$ (by rules s1, s2, s5).
C7. $\Gamma \vdash Dest_B(B, m_3)$ (by rule d).
C8. $\Gamma \vdash PIPE(A, \langle N_B + 1, m_3 \rangle, B)$ (by pipe.rule).

A similar proof shows that $\Gamma_{Andrew} \vdash G_2$ and $\Gamma_{Andrew} \vdash G_3$

- **Resistance against known attacks:** Against Clark-Jacob [45] attack.
- **Code for verification using Scyther**

```
usertype SessionKey;
const succ: Function;
const Fresh: Function;
const sender: Function;
const reciever: Function;

protocol andrew(A,B)
{
   role A
   {
      fresh NA: Nonce;
      var NB,NB': Nonce;
      var k'AB: SessionKey;

      send_1(A,B, {A, B, NA}k(A,B) );
    recv_2(B,A, {B, A, succ(NA),NB}k(A,B) );
      send_3(A,B, {A, B, succ(NB)}k(A,B) );
    recv_4(B,A, {B, A, k'AB,NB', NA}k(A,B) );
      claim_A1(A,Nisynch);
   }

   role B
   {
      var NA: Nonce;
      fresh NB,NB': Nonce;
      fresh k'AB: SessionKey;

      recv_1(A,B,{A, B, NA}k(A,B) );
    send_2(B,A, {B, A, succ(NA),NB}k(A,B) );
      recv_3(A,B, {A, B, succ(NB)}k(A,B) );
    send_4(B,A, {B, A, k'AB,NB', NA}k(A,B) );
      claim_B1(B,Nisynch);
   }
}
```

- **Result of the simulation**

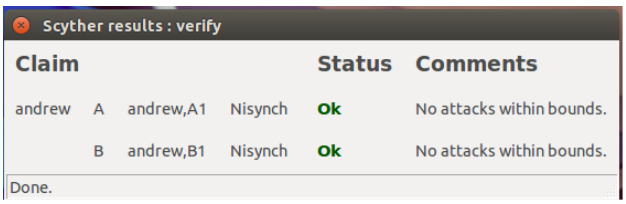

Figure 5. Simulation results for Andrew Secure RPC protocol.

7.3 Revised Andrew Protocol of Burrows

The Revised Andrew protocol [25].

- **Informal Description of the Protocol**:

1. $A \mapsto B : m_1 = \uparrow_A, N_A$
2. $B \mapsto A : m_2 = \{N_A, K'_{AB}\}_{K_{AB}}$
3. $A \mapsto B : m_3 = \{N_A\}_{K'_{AB}}$
4. $B \mapsto A : m_4 = \{N'_B\}_{K_{AB}}$

Protocol 15. Revised Andrew protocol of Burrows *et al.*

- **Formal Model of the Protocol:** the set of assumptions of the Revised Andrew protocol is

$$\Gamma_{RA} = \{Hid(K_{AB}), Hid(K'_{AB}), Fresh_A(N_A), Fresh_B(N'_B)\}.$$

Goals of the protocol are

$$G_1 = PIPE(B, \langle K'_{AB}, m_2 \rangle, A),$$

and

$$G_2 = PIPE(A, \langle N_A, m_3 \rangle, B).$$

- **Verification:** It is easily seen that $\Gamma_{RA} \nvdash G_1$, and $\Gamma_{RA} \nvdash G_2$.
- **Modification:** We modify the structure of messages of Revised Andrew protocol such that goals G_1 and G_2 can be derived from. Γ_{RA}.

1. $A \mapsto B : m_1 = \{tuple_3(\uparrow_A, \downarrow_B, N_A)\}_{K_{AB}}$
2. $B \mapsto A : m_2 = \{tuple_4(\uparrow_B, \downarrow_A, N_A, K'_{AB})\}_{K_{AB}}$
3. $A \mapsto B : m_3 = \{tuple_3(\uparrow_A, \downarrow_B, N_A)\}_{K'_{AB}}$
4. $B \mapsto A : m_4 = \{tuple_3(\uparrow_B, \downarrow_A, N'_B)\}_{K_{AB}}$

Protocol 16. Modified revised Andrew protocol of Burrows *et al.*

Now we prove that $\Gamma_{RA} \vdash G_1$ and $\Gamma_{RA} \vdash G_2$.

- **Resistance against known attacks:** Mirror attack by Lowe.
- **Code for verification using Scyther**

```
usertype SessionKey;
#const Fresh: Function;

protocol andrew-revised(A,B)
```

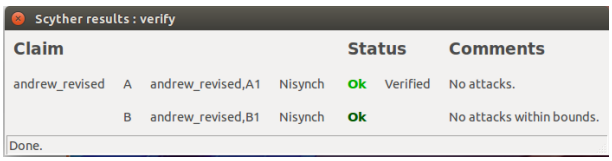

Figure 6. Simulation results for revised Andrew protocol of Burrows.

```
{
  role A
  {
    fresh NA: Nonce;
    var NB,NB': Nonce;
    var k'AB: SessionKey;

    send_1(A,B, {A,B,NA}k(A,B) );
    recv_2(B,A, {B,A,NA,k'AB}k(A,B) );
    send_3(A,B, {A,B,NA}k'AB );
    claim_A1(A,Nisynch);
recv_4(B,A, {B,A,NB'}k(A,B) );
  }

  role B
  {
    var NA: Nonce;
    fresh NB,NB': Nonce;
    fresh k'AB: SessionKey;

    recv_1(A,B, {A,B,NA}k(A,B) );
    send_2(B,A, {B,A,NA,k'AB}k(A,B) );
    recv_3(A,B, {A,B,NA}k'AB );
    send_4(B,A, {B,A,NB'}k(A,B) );
    claim_B1(B,Nisynch);
  }
}
```

- **Result of the simulation**

7.4 ISO/IEC 9798-2 Protocol

The ISO/IEC 9798-2 provides mutual authentication [46].

- **Informal Description of the Protocol**:

1. $B \mapsto A : m_1 = N_B$
2. $A \mapsto B : m_2 = \{N_A, N_B, \downarrow_B\}_{K_{AB}}$
3. $B \mapsto A : m_3 = \{N_B, N_A\}_{K_{AB}}$

Protocol 17. ISO/IEC 9798-2 three-pass mutual authentication protocol.

- **Formal Model of the Protocol:** the set of assumptions of the ISO/IEC 9798-2 protocol is

$$\Gamma_{ISO} = \{Hid(K_{AB}), Fresh_A(N_A), Fresh_B(N_B)\}.$$

Goals of the protocol are

$$G_1 = PIPE(A, \langle N_B, m_2\rangle, B),$$

and

$$G_2 = PIPE(B, \langle N_A, m_3\rangle, A).$$

- **Verification:** It is easily seen that $\Gamma_{ISO} \nvdash G_1$ and $\Gamma_{ISO} \nvdash G_2$.
- **Modification:** We modify the structure of messages of ISO/IEC 9798-2 protocol such that goals G_1 and G_2 can be derived from Γ_{ISO}.

1. $B \mapsto A : m_1 = tuple_1(N_B)$
2. $A \mapsto B : m_2 = \{tuple_4(\uparrow_A, \downarrow_B, N_A, N_B)\}_{K_{AB}}$
3. $B \mapsto A : m_3 = \{tuple_4(\uparrow_B, \downarrow_A, N_B, N_A)\}_{K_{AB}}$

Protocol 18. Modified ISO/IEC 9798-2 protocol.

Now we prove that $\Gamma_{ISO} \vdash G_1$.

C1. $\Gamma \vdash Fresh_B(m_2)$ (by rules f1, f4 and assumption $Fresh_B(N_B)$).
C2. $\Gamma \vdash Integ_B(m_2)$ (by rule i2 and assumption $Hid(K_{AB})$).
C3. $\Gamma \vdash Const_B(A, m_2)$ (by rule c3 and assumption $Hid(K_{AB})$).
C4. $\Gamma \vdash Sub_B(\uparrow_B, m_2)$ (by rules s1, s2, s5).
C5. $\Gamma \vdash Source_B(A, m_2)$ (by rules so, C3 and C4).
C6. $\Gamma \vdash Sub_B(N_B, m_2)$ (by rules s1, s2, s5).
C7. $\Gamma \vdash Dest_B(B, m_2)$ (by rule d).
C8. $\Gamma \vdash PIPE(A, \langle N_B, m_2\rangle, B)$ (by pipe.rule).

A similar proof shows that $\Gamma_{ISO} \vdash G_2$.

- **Resistance against known attacks:**

In some especial environments the receiver of the message can not determine who sent the first message, so is obvious to use the text field $Text1$ to indicate this, so the protocol becomes:

1. $B \mapsto A : m_1 = N_B, B$
2. $A \mapsto B : m_2 = \{N_A, N_B, \downarrow_B\}_{K_{AB}}$
3. $B \mapsto A : m_3 = \{N_B, N_A\}_{K_{AB}}$

Protocol 19. ISO/IEC 9798-2 three-pass mutual authentication protocol version 2.

In such case, an intruder E starts a second protocol run, impersonating A to B, with sending message (N_B, B). B accepts this and replies with the appropriate response $\{N'_B, N_B, \downarrow_B\}_{K_{AB}}$ and then proceeds the first protocol run impersonating A. In our modified version this attack is not possible since, when B, replies the first message with $\{tuple_4(\uparrow_B, \downarrow_A, N'_B, N_B)\}_{K_{AB}}$ and

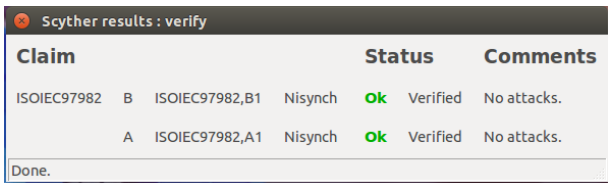

Figure 7. Simulation results for ISO/IEC 9798-2 protocol.

therefore the intruder can not use this message in the first run.

- **Code for verification using Scyther**

```
usertype SessionKey;
#const Fresh: Function;

protocol andrew-LoweBan(B,A)
{
   role B
   {
      fresh NB: Nonce;
      var NA: Nonce;

      send_1(B,A, NB );
      recv_2(A,B, {A,B,NA,NB}k(A,B) );
      send_3(B,A, {B,A,NB,NA}k(A,B) );
      claim_B1(B,Nisynch);
   }

   role A
   {
      var NB: Nonce;
      fresh NA: Nonce;

      recv_1(B,A, NB );
      send_2(A,B, {A,B,NA,NB}k(A,B) );
      recv_3(B,A, {B,A,NB,NA}k(A,B) );
      claim_A1(A,Nisynch);
   }
}
```

- **Result of the simulation**

7.5 ISO/IEC 11770-2

The ISO/IEC 11770-2 is a Key Establishment protocol [47].

- **Informal Description of the Protocol**:

1. $B \mapsto A : m_1 = N_B$
2. $A \mapsto B : m_2 = \{N_B, \downarrow_B, K'_{AB}\}_{K_{AB}}$

Protocol 20. ISO/IEC 11770-2 key establishment.

- **Formal Model of the Protocol:** the set of assumptions of the ISO/IEC 11770-2 protocol is

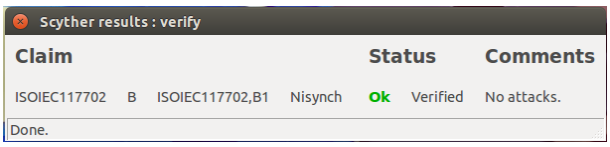

Figure 8. Simulation results for ISO/IEC 11770-2 protocol.

$$\Gamma_{ISO} = \{Hid(K_{AB}), Hid(K'_{AB}), Fresh_B(N_B)\}.$$

Goal of the protocol is

$$G_1 = PIPE(A, \langle K'_{AB}, m_2\rangle, B),$$

and

- **Verification:** It is easily seen that $\Gamma_{ISO} \nvdash G_1$.
- **Modification:** We modify the structure of messages of ISO/IEC 11770-2 protocol such that goal G_1 can be derived from. Γ_{ISO}.

1. $B \mapsto A : m_1 = tuple_1(N_B)$
2. $A \mapsto B : m_2 = \{tuple_4(\uparrow_A, \downarrow_B, N_B, K'_{AB})\}_{K_{AB}}$

Protocol 21. Modified ISO/IEC 11770-2 key establishment.

Now we prove that $\Gamma_{ISO} \vdash G_1$.

- **Resistance against known attacks:** SG logic
- **Code for verification using Scyther**

```
usertype SessionKey;
#const Fresh: Function;

protocol ISOIEC117702(B,A)
{
   role B
   {
      fresh NB: Nonce;
      var NA: Nonce;
      var k'AB: SessionKey;

      send_1(B,A, NB );
      recv_2(A,B, {A,B,NB,k'AB}k(A,B) );
      claim_B1(B,Nisynch);
   }

   role A
   {
      var NB: Nonce;
      fresh NA: Nonce;
fresh k'AB: SessionKey;

      recv_1(B,A, NB );
      send_2(A,B, {A,B,NB,k'AB}k(A,B) );
   }
}
```

- **Result of the simulation**

7.6 ISO/IEC 9798-3 Protocol

The ISO/IEC 9798-3 protocol [48].

- **Informal Description of the Protocol**:

1. $B \mapsto A : m_1 = N_B$
2. $A \mapsto B : m_2 = N_A, N_B, \downarrow_B, \{N_A, N_B, \downarrow_B\}_{sk_A}$
3. $B \mapsto A : m_3 = N'_B, N_A, \downarrow_A, \{N'_B, N_A, \downarrow_A\}_{sk_B}$

Protocol 22. ISO/IEC 9798-3 three-pass mutual authentication.

- **Formal Model of the Protocol:** the set of assumptions of the ISO/IEC 9798-3 protocol is

$$\Gamma_{ISO} = \{Hid(sk_B), Hid(sk_A), Fresh_A(N_A), Fresh_B(N_B)\}.$$

Goals of the protocol are

$$G_1 = PIPE(A, \langle N_B, m_2\rangle, B),$$

and

$$G_2 = PIPE(B, \langle N_A, m_3\rangle, A).$$

- **Verification:** It is easily seen that $\Gamma_{ISO} \nvdash G_1$ and $\Gamma_{ISO} \nvdash G_2$.
- **Modification:** We modify the structure of messages of ISO/IEC 9798-3 protocol such that goals G_1 and G_2 can be derived from Γ_{ISO}.

1. $B \mapsto A : m_1 = tuple_1(N_B)$
2. $A \mapsto B : m_2 = tuple_2(tuple_3(N_A, N_B, \downarrow_B), \{tuple_4(\uparrow_A, \downarrow_B N_A, N_B)\}_{sk_A})$
3. $B \mapsto A : m_3 = tuple_2(tuple_3(N'_B, N_A, \downarrow_A), \{tuple_4(\uparrow_B, \downarrow_A, N'_B, N_A)\}_{sk_B})$

Protocol 23. Modified ISO/IEC 9798-3 three-pass mutual authentication.

Now we prove that $\Gamma_{ISO} \vdash G_1$ and $\Gamma_{ISO} \vdash G_2$.

C1. $\Gamma \vdash Fresh_B(m_2)$ (by rules f1, f2 and assumption $Fresh_B(N_B)$).
C2. $\Gamma \vdash Integ_B(m_2)$ (by rule i3 and assumption $Hid(sk_A)$).
C3. $\Gamma \vdash Const_B(A, m_2)$ (by rule c5 and assumption $Hid(sk_A)$).
C4. $\Gamma \vdash Sub_B(\uparrow_A, m_2)$ (by rules s1,s3,s5).
C5. $\Gamma \vdash Source_B(A, m_2)$ (by rules so, C3 and C4).
C6. $\Gamma \vdash Sub_B(N_B, m_2)$ (by rules s1, s3, s5).
C7. $\Gamma \vdash Dest_B(B, m_2)$ (by rule d).
C8. $\Gamma \vdash PIPE(A, \langle N_B, m_2\rangle, B)$ (by pipe.rule).

A similar proof shows that $\Gamma_{ISO} \vdash G_2$.

7.7 ISO/IEC 11770-3 Protocol

The ISO/IEC 11770-3 protocol [49].

- **Informal Description of the Protocol**:

1. $A \mapsto B : m_1 = N_A$
2. $B \mapsto A : m_2 = \downarrow_A, N_A, N_B, \{\uparrow_B, K_{AB}\}_{pk_A}, \{\downarrow_A, N_A, N_B, \{\uparrow_B, K_{AB}\}_{pk_A}\}_{sk_B}$

Protocol 24. ISO/IEC 11770-3 key transport.

- **Formal Model of the Protocol:** the set of assumptions of the ISO/IEC 11770-3 protocol is

$$\Gamma_{ISO} = \{Hid(sk_B), Fresh_A(N_A), Fresh_B(N_B)\}.$$

Goal of the protocols are

$$G = PIPE(B, \langle K_{AB}, m_2\rangle, A).$$

- **Verification:** We show that $\Gamma_{ISO} \vdash G$

C1. $\Gamma \vdash Fresh_A(m_2)$ (by rules f1, f2 and assumption $Fresh_A(N_A)$).
C2. $\Gamma \vdash Integ_A(m_2)$ (by rule i3 and assumption $Hid(sk_B)$).
C3. $\Gamma \vdash Const_A(B, m_2)$ (by rule c5 and assumption $Hid(sk_B)$).
C4. $\Gamma \vdash Sub_A(\uparrow_B, m_2)$ (by rules s1, s3, s4, s5).
C5. $\Gamma \vdash Source_A(B, m_2)$ (by rules so, C3 and C4).
C6. $\Gamma \vdash Sub_A(K_{AB}, m_2)$ (by rules s1, s3, s4, s5).
C7. $\Gamma \vdash Dest_A(A, m_2)$ (by rule d).
C8. $\Gamma \vdash PIPE(B, \langle K_{AB}, m_2\rangle, A)$ (by pipe.rule).

- **Code for verification using Scyther**

```
usertype SessionKey;
#const Fresh: Function;

protocol ISOIEC117703(B,A)
{
   role A
   {
      fresh NA: Nonce;
      var NB: Nonce;
      var kAB: SessionKey;

      send_1(A,B, NA );
      recv_2(B,A, NA,NB,{B,kAB}pk(A),{A,
       NA,NB,{B,kAB}pk(A)}sk(B));
      claim_A1(A,Nisynch);
}

   role B
   {
      fresh NB: Nonce;
      fresh NA: Nonce;
      fresh kAB: SessionKey;

      recv_1(A,B, NA );
      send_2(B,A, NA,NB,{B,kAB}pk(A),
```

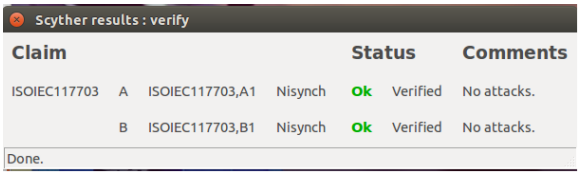

Figure 9. Simulation results for ISO/IEC 11770-3 protocol.

```
        {A,NA,NB,{B,kAB}pk(A)}sk(B) );
      claim_B1(B,Nisynch);
    }
}
```

- **Result of the simulation**

8 Conclusion

Different protocols are used for different purposes. The most important aspect of every protocol is that whether they satisfy the required purposes or not. There are many instances of protocols in the literature that were assumed to be secure for many years, but they had serious flows. Verifying the correctness of the protocols are a very tedious task. In order to overcome this, automatic methods are used. The most famous approaches in this regard are model checking and theorem proving. In model checking different behavior of a protocol is modeled as a graph and the different paths show different behavior of the protocol. Theorem proving tries to proof the properties of the protocol by logical reasoning about different implications of a protocol.

In this paper we first discuss different security attacks and goals, and introduced the *Scyther* as one of the latest model checking tools. After that, we provided an intuition about a new theorem proving named *Simple Logic for Authentication*, which tries to reason about a protocol based on the structure of its messages.

References

[1] Roger M Needham and Michael D Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[2] Dorothy E Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[3] Stuart G Stubblebine and Virgil D Gligor. *On message integrity in cryptographic protocols*. IEEE, 1992.

[4] Wenbo Mao and Colin Boyd. On the use of encryption in cryptographic protocols. *Codes and Cyphers: Cryptography and Coding IV*, pages 251–262, 1995.

[5] Paul Syverson. A taxonomy of replay attacks [cryptographic protocols]. In *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings*, pages 187–191. IEEE, 1994.

[6] Mike Burmester. On the risk of opening distributed keys. In *Advances in Cryptology-CRYPTO94*, pages 308–317. Springer, 1994.

[7] Colin Boyd and Anish Mathuria. *Protocols for authentication and key establishment*. Springer Science & Business Media, 2013.

[8] Ray Bird, Inder Gopal, Amir Herzberg, Philippe Janson, Shay Kutten, Refik Molva, Moti Yung, et al. Systematic design of a family of attack-resistant authentication protocols. *Selected Areas in Communications, IEEE Journal on*, 11(5):679–693, 1993.

[9] Tuomas Aura and Pekka Nikander. Stateless connections. *Information and Communications Security*, pages 87–97, 1997.

[10] Catherine Meadows. A formal framework and evaluation method for network denial of service. In *Computer Security Foundations Workshop, 1999. Proceedings of the 12th IEEE*, pages 4–13. IEEE, 1999.

[11] Ari Juels and John G Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, volume 99, pages 151–165, 1999.

[12] Dave Otway and Owen Rees. Efficient and timely mutual authentication. *ACM SIGOPS Operating Systems Review*, 21(1):8–10, 1987.

[13] Casimier Joseph Franciscus Cremers. *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology, 2006.

[14] Cas Cremers and Sjouke Mauw. *Operational semantics and verification of security protocols*. Springer Science & Business Media, 2012.

[15] Markus Müller-Olm, David Schmidt, and Bernhard Steffen. Model-checking. In *Static Analysis*, pages 330–354. Springer, 1999.

[16] Edmund M Clarke and E Allen Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*. Springer, 1982.

[17] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on Programming*, pages 337–351. Springer, 1982.

[18] Amir Pnueli. The temporal semantics of concurrent programs. *Theoretical computer science*, 13(1):45–60, 1981.

[19] David L Dill. The mur ϕ verification system. In *Computer Aided Verification*, pages 390–393.

Springer, 1996.

[20] John C Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using murφ. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*, pages 141–151. IEEE, 1997.

[21] Makoto Tatebayashi, Natsume Matsuzaki, and David B Newman Jr. Key distribution protocol for digital mobile communication systems. In *Advances in CryptologyCRYPTO89 Proceedings*, pages 324–334. Springer, 1990.

[22] B Clifford Neuman and Theodore Ts' O. Kerberos: An authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, 1994.

[23] Edmund M Clarke, Somesh Jha, and Will Marrero. Verifying security protocols with brutus. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(4):443–487, 2000.

[24] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification*, pages 414–418. Springer, 2008.

[25] Michael Burrows, Martin Abadi, and Roger M Needham. A logic of authentication. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 426, pages 233–271. The Royal Society, 1989.

[26] Michael Burrows, Martin Abadi, and Roger M Needham. A logic of cryptographic. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[27] Stefanos Gritzalis, Diomidis Spinellis, and Panagiotis Georgiadis. Security protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification. *Computer Communications*, 22(8):697–709, 1999.

[28] Recommendation X CCITT. 509: The directory authentication framework, 1988.

[29] Steven P Miller, B Clifford Neuman, Jeffrey I Schiller, and Jermoe H Saltzer. Kerberos authentication and authorization system. In *In Project Athena Technical Plan*. Citeseer, 1987.

[30] Dan M Nessett. A critique of the burrows, abadi and needham logic. *ACM SIGOPS Operating Systems Review*, 24(2):35–38, 1990.

[31] Paul Syverson. The use of logic in the analysis of cryptographic protocols. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 156–170. IEEE, 1991.

[32] Wenbo Mao and Colin Boyd. Towards formal analysis of security protocols. In *Computer Security Foundations Workshop VI, 1993. Proceedings*, pages 147–158. IEEE, 1993.

[33] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*, pages 234–248. IEEE, 1990.

[34] Martin Abadi and Mark R Tuttle. A semantics for a logic of authentication. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 201–216. ACM, 1991.

[35] Paul F Syverson and Paul C Van Oorschot. On unifying some cryptographic protocol logics. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 14–28. IEEE, 1994.

[36] Volker Kessler and Gabriele Wedel. Autlog-an advanced logic of authentication. In *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings*, pages 90–99. IEEE, 1994.

[37] Paul van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 232–243. ACM, 1993.

[38] ITU-TS, Recommendation Z.120: Message Sequence Chart (MSC) ITU-TS, Geneva. 1999.

[39] Danny Dolev and Andrew C Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[40] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[41] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166. Springer, 1996.

[42] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in CryptologyCRYPTO93*, pages 232–249. Springer, 1994.

[43] Jim Alves-Foss. Provably insecure mutual authentication protocols: The two party symmetric encryption case. In *Proc. 22nd National Information Systems Security Conference*, pages 44–55. Citeseer, 1999.

[44] Mahadev Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems (TOCS)*, 7(3):247–280, 1989.

[45] John Clark and Jeremy Jacob. Attacking authentication protocols. *High Integrity Systems*, 1:465–474, 1996.

[46] *ISO. Information Technology - Security Tech-*

niques - Entity Authentication - Part 2: Mechanisms Using Symmetric Encipherment Algorithms ISO/IEC 9798-2, 2nd edition, 1999, International Standard.

[47] *ISO. Information Technology - Security Techniques - Key Management - Part 2: Mechanisms Using Symmetric Techniques ISO/IEC11770-2*, 2nd edition, 1996, International Standard.

[48] *ISO. Information Technology - Security Techniques - Entity Authentication Mechanisms - Part 3: Entity Authentication Using a Public Key Algorithm ISO/IEC 9798-3*, 2nd edition, 1998, International Standard.

[49] *ISO. Information Technology - Security Techniques - Key Management - Part 3: Mechanisms Using Asymmetric Techniques ISO/IEC 11770-3*, 2nd edition, 1999, International Standard.

Mohsen Pourpouneh was born in 1989 in Isfahan. He got his B.Sc. (2011) and M.Sc. (2013) in Computer Science, from Shahid Beheshti University and Tehran University, respectively. He started his career as a Ph.D. student at Sharif University of Technology, Tehran, Iran in 2013. His research interest includes formal method, electronic voting protocols, and multi-agent systems.

Rasoul Ramezanian was born in Mashhad in 1979. He got his B.S. and M.S. in Mathematics. In 2008, he graduated from a Ph.D. program of Mathematical Science Department of Sharif University of Technology, Tehran, Iran. He is an assistant professor at the faculty of Mathematical Sciences at Ferdowsi University of Mashhad. His research interests include formal method, specifying and verifying security protocols, multi-agent systems, and process algebra.