Equational Approach to Formal Analysis of TLS

Kazuhiro Ogata
NEC Software Hokuriku, Ltd.
ogatak@acm.org

Kokichi Futatsugi
School of Information Science, JAIST
kokichi@jaist.ac.jp

Abstract

TLS has been formally analyzed with the OTS/CafeOBJ method. In the method, distributed systems are modeled as transition systems, which are written in terms of equations, and it is verified that the models have properties by means of equational reasoning. TLS is the latest version, or the successor of SSL, which is probably the most widely deployed security protocol. Among the results of the analysis are that pre-master secrets cannot be leaked, when a client has negotiated a cipher suite and security parameters with a server, the server has really agreed on them, and client cannot be identified if they do not send their certificates to servers.

Keywords: algebraic specification, interactive theorem proving, security, rewriting, verification.

1. Introduction

A large number of security protocols have been proposed so as to protect data at the application and transport layers. They are extremely essential in the Internet era. The most widely deployed security protocol among them is most likely SSL (Secure Sockets Layer). Most of the WWW browsers adopt SSL. Their users send their personal information such as credit card numbers to purchase goods from electronic malls. It is then really important that the security protocol is truly secure. We have formally analyzed TLS[2] (Transport Layer Security), which is the latest version, or the successor of SSL, so as to confirm that it is secure.

The OTS/CafeOBJ method[9] has been used to analyze the protocol. CafeOBJ[1] is an algebraic specification language/system, with which software/hardware systems are specified in terms of equations and it is verified that the systems have properties by means of equational reasoning. OTSs (observational transition systems) are transition systems suited for being written in equations. A security protocol, together with the most general intruder, is modeled as an OTS. The OTS is described in equations with CafeOBJ. Properties to analyze are expressed as CafeOBJ terms, and proof scores that the OTS has the properties are also written in CafeOBJ. The proof scores are then verified with the rewriting facilities of CafeOBJ. Rewriting is an efficient way of implementing equational reasoning.

In the current OTS/CafeOBJ method, basically proof scores should be written by hand, although most part of a proof score that an OTS has a property can be reused for other proof scores that the OTS has other properties. But, it took about one week to verify 18 invariants in the case study described in the paper by writing proof scores in CafeOBJ. This is most likely because the basic proof method used in the case study, namely equational reasoning using rewriting, can moderate the difficulties of proofs that might otherwise become too hard to understand. This also does not show that interactive theorem proving costs high as often said notoriously.

In this paper, we describe how TLS has been analyzed. Among the results of the analysis are that pre-master secrets cannot be leaked, when a client has negotiated a cipher suite and security parameters with a server, the server has really agreed on them, and client cannot be identified if they dot not send their certificates to servers.

The rest of the paper is organized as follows. Section 2 mentions the OTS/CafeOBJ method. Section 3 describes TLS and presents the abstract handshake protocol that has been analyzed. The way of analyzing the abstract handshake protocol is described in Section 4 and Section 5. Section 6 discusses some related work, and we conclude the paper in Section 7.

2. The OTS/CafeOBJ Method

2.1. CafeOBJ: Algebraic Specification Language

Abstract machines as well as abstract data types can be specified in CafeOBJ[1][1], which are mainly based on hidden and initial algebras. CafeOBJ has two kinds of sorts: visible and hidden sorts denoting abstract data types and the state spaces of abstract machines. CafeOBJ has two

[1] See `www.ldl.jaist.ac.jp/cafeobj/`.

kinds of operators wrt (with respect to) hidden sorts: action and observation operators that denote state transitions of abstract machines and let us know the situation where abstract machines are located. Both an action operator and an observation operator take a state of an abstract machine and zero or more data, and return the successor state of the state and a value that characterizes the situation of the abstract machine.

Action and observation operators are declared by starting with `bop`, and others by starting with `op`. After `bop` or `op`, an operator name is written, followed by `:` and a list of sorts, and then, `->` and a sort are written. Operators are defined with equations. Equations are declared by starting with `eq`, and conditional ones by starting with `ceq`. After `eq`, two terms connected with `=` are written, ended with a full stop. After `ceq`, two terms connected with `=` are written, followed by `if`, and then, a term denoting the condition and a full stop are written. The CafeOBJ system uses equations as left-to-right rewrite rules and rewrites a given term. The command `red` is used to rewrite a given term.

Basic units of CafeOBJ specifications are modules. The CafeOBJ system provides built-in modules such as `BOOL` where propositional logic is specified. The import of `BOOL` lets us use the visible sort `Bool` denoting truth values, the constants `true` and `false`, and some logical operators such as `not_` (negation), `_and_` (conjunction) and `_implies_` (implication). The operator `if_then_else_fi` (choice) is also available. An underscore `_` indicates the place where an argument is put.

`BOOL` plays an essential role in verification with the CafeOBJ system. If the equations available in the module are regarded as left-to-right rewrite rules, they are complete wrt propositional logic[5]. Any term denoting a propositional formula that is always true (or false) is surely rewritten to `true` (or `false`).

2.2. Observational Transition Systems (OTSs)

We assume that there exists a universal state space denoted by Υ. We also assume that data types used, including the equivalence relation (denoted by $=$) for each data type, have been defined in advance.

An OTS[9] $\mathcal{S}$ consists of $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ where

- $\mathcal{O}$: A set of observers. Each $o \in \mathcal{O}$ is a function $o : \Upsilon \rightarrow D$, where D is a data type and may differ from observer to observer. Given two states $v_1, v_2 \in \Upsilon$, the equivalence (denoted by $v_1 =_{\mathcal{S}} v_2$) between them wrt $\mathcal{S}$ is defined as $\forall o \in \mathcal{O}.o(v_1) = o(v_2)$.
- $\mathcal{I}$: The set of initial states such that $\mathcal{I} \subseteq \Upsilon$.
- $\mathcal{T}$: A set of conditional transitions. Each $\tau \in \mathcal{T}$ is a function $\tau : \Upsilon \rightarrow \Upsilon$ such that $\tau(v_1) =_{\mathcal{S}} \tau(v_2)$ for each $[v] \in \Upsilon/=_{\mathcal{S}}$ and each $v_1, v_2 \in [v]$. $\tau(v)$ is called the successor state of $v \in \Upsilon$ wrt τ. The condition c_τ of τ is called the effective condition. For each $v \in \Upsilon$ such that $\neg c_\tau(v)$, $v =_{\mathcal{S}} \tau(v)$.

An execution of $\mathcal{S}$ is an infinite sequence $v_0, v_1, \ldots$ of states satisfying Initiation ($v_0 \in \mathcal{I}$) and Consecution (for each $i \in \{0, 1, \ldots\}$, $v_{i+1} =_{\mathcal{S}} \tau(v_i)$ for some $\tau \in \mathcal{T}$). A state v is called reachable wrt $\mathcal{S}$ iff (if and only if) there exists an execution of $\mathcal{S}$ in which v appears. Let $\mathcal{R}_{\mathcal{S}}$ be the set of all reachable states wrt $\mathcal{S}$. All properties considered in this paper are invariants. A predicate p is called invariant wrt $\mathcal{S}$ iff $\forall v \in \mathcal{R}_{\mathcal{S}}.p(v)$.

Observers and transitions may be parameterized. Observers and transitions are generally expressed as $o_{i_1,\ldots,i_m}$ and $\tau_{j_1,\ldots,j_n}$, provided that $m, n \geq 0$ and there exists a data type D_k such that $k \in D_k$ ($k = i_1, \ldots, i_m, j_1, \ldots, j_n$).

2.3. Specification of OTSs in CafeOBJ

Υ is denoted by a hidden sort, say *H*, $o_{i_1,\ldots,i_m}$ by a CafeOBJ observation operator, say *o*, and $\tau_{j_1,\ldots,j_n}$ by a CafeOBJ action operator, say *a*. An action operator is basically specified with equations by describing how the value returned by each observation operator changes. A typical form of such equations looks like

$$\texttt{ceq}\ o(a(S, X_{j_1}, \ldots, X_{j_n}), X_{i_1}, \ldots, X_{i_m}) = e\text{-}a(S, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m})\ \texttt{if}\ c\text{-}a(S, X_{j_1}, \ldots, X_{j_n})\ .$$

S is a CafeOBJ variable of *H* and each X_k is a CafeOBJ variable of the visible sort denoting D_k. $a(S, X_{j_1}, \ldots, X_{j_n})$ denotes the successor state of *S* wrt $\tau_{j_1,\ldots,j_n}$. $e\text{-}a(S, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m})$ denotes the value returned by $o_{i_1,\ldots,i_m}$ in the successor state. The effective condition $c_{\tau_{j_1,\ldots,j_n}}$ is denoted by $c\text{-}a(S, X_{j_1}, \ldots, X_{j_n})$.

2.4. Verification of OTSs

Some invariants may be proved by case analysis only. But, we often need induction, especially simultaneous induction on the number of transitions applied. We then describe how to prove a predicate p_1 invariant to $\mathcal{S}$ by such induction by writing proof scores in CafeOBJ[9]. The proof that p_1 is invariant to $\mathcal{S}$ often needs other predicates. We suppose that $p_2, \ldots, p_n$ are such predicates. We then prove $p_1 \wedge \ldots \wedge p_n$ invariant to $\mathcal{S}$, but each proof that p_i ($i = 1, \ldots, n$) is invariant to $\mathcal{S}$ is written individually. Let $x_{i1}, \ldots, x_{im_i}$ whose types are $D_{i1}, \ldots, D_{im_i}$ be all free variables in p_i ($i = 1, \ldots, n$) except for v whose type is Υ.

We first declare the operators denoting $p_1, \ldots, p_n$ and their defining equations in a module `INV` (which imports the module where $\mathcal{S}$ is written) as follows:

$$\texttt{op}\ inv_i : H\ V_{i1} \ldots V_{im_i}\ \texttt{->}\ \texttt{Bool}$$
$$\texttt{eq}\ inv_i(S, X_{i1}, \ldots, X_{im_i}) = p_i(S, X_{i1}, \ldots, X_{im_i})\ .$$

where $i = 1, \ldots, n$. V_k $(k = i1, \ldots, im_i)$ is a visible sort denoting D_k, and X_k is a CafeOBJ variable whose sort is V_k. $p_i(S, X_{i1}, \ldots, X_{im_i})$ is a CafeOBJ term denoting p_i. In `INV`, we also declare a constant x_k denoting an arbitrary value of V_k $(k = 1, \ldots, n)$. We then declare the operators denoting basic formulas to show in the inductive cases and their defining equations in a module `ISTEP` (which imports `INV`) as follows:

```
op istep_i : V_i1 ... V_im_i -> Bool
eq istep_i(X_i1, ..., X_im_i)
   = inv_i(s, X_i1, ..., X_im_i) implies inv_i(s', X_i1, ..., X_im_i) .
```

where $i = 1, \ldots, n$. s and s' are constants of *H*, denoting an arbitrary state and a successor state of *s*.

The proof of each inductive case often needs case analysis. Let us consider the inductive case where it is shown that $\tau_{j_1,\ldots,j_{m_j}}$ preserves p_i. Suppose that the state space is split into l sub-spaces for the proof of the inductive case and each sub-space is characterized by a predicate $case_{i_k}$ $(k = 1, \ldots, l)$ such that $(case_{i_1} \vee \ldots \vee case_{i_l}) \Leftrightarrow \mathrm{true}$. Also suppose that $\tau_{j_1,\ldots,j_{m_j}}$ is denoted by a CafeOBJ action operator *a* and visible sorts $V_{j_1}, \ldots, V_{j_{m_j}}$ correspond to data types $D_{j_1}, \ldots, D_{j_{m_j}}$ of the parameters of $\tau_{j_1,\ldots,j_{m_j}}$. Then the proof looks like

```
open ISTEP
 -- arbitrary objects
 op y_1m_1 :-> V_1m_1 . ... op y_jm_j :-> V_jm_j .
 -- assumptions
 Declaration of equations denoting case_i_k .
 -- successor state
 eq s' = a(s, y_j1, ..., y_jm_j) .
 -- check
 red SIH_i implies istep_i(x_i1, ..., x_im_i) .
close
```

where $i = 1, \ldots, n$ and $k = 1, \ldots, l$. A comment starts with `--` and terminates at the end of the line. SIH_i is a CafeOBJ term denoting what strengthens the inductive hypothesis $inv_i(s, x_{i1}, \ldots, x_{im_i})$, and can be $inv_{i^1}(s, t_{i^1 1}, \ldots, t_{i^1 m_{i^1}})$ and ... and $inv_{i^{u_i}}(s, t_{i^{u_i} 1}, \ldots, t_{i^{u_i} m_{i^{u_i}}})$ where $i^\imath \in \{1, \ldots, n\}$, $t_{i^\imath \jmath}$ is a term whose sort is $V_{i^\imath \jmath}$, $\imath = 1, \ldots, u_i$ and $\jmath = 1, \ldots, m_{i^\imath}$. `open` makes a temporary module that imports a module given as an argument, and `close` destroys the temporary module. Parts enclosed with `open` and `close` are basic units of proof scores, which are called proof passages.

3. TLS

TLS[2] is a general protocol that can be used to secure any exchanges between two points and the successor of SSL, which was integrated in 1994 in Netscape Navigator.

TLS consists of four subprotocols, which are the TLS handshake protocol, the change cipher spec protocol, the alert protocol and the TLS record protocol. Two peers use the TLS handshake protocol to negotiate a cipher suite and security parameters. Each of two peers uses the change cipher spec protocol to notify the other that subsequent messages will be protected under the newly negotiated cipher suite and security parameters. If either of two peers notices something wrong, he/she lets the other know about it with the alert protocol. The TLS record protocol secures communications between the two peers using the negotiated cipher suite and security parameters.

The security of TLS crucially depends on a cipher suite and security parameter negotiated by two peers. If a cipher suite used is weaker then one that is available to both peers or security parameters shared by both peers are leaked, then an adequate level of security cannot be obtained. Therefore, the security of TLS largely relies on that of the TLS handshake protocol.

3.1. The TLS Handshake Protocol

Messages exchanged in the TLS handshake protocol are depicted in Figure 1. Messages marked by $*$ are optional, and those surrounded by square brackets are for the change cipher spec protocol.

A server may send a HelloRequest message to a client to initiate a new run of the protocol. The client replies to the HelloRequest message with a ClientHello message or sends a ClientHello message to the server to initiate a new run of the protocol. A ClientHello message consists of a version number of TLS, a list of cipher suites that are available to a client, a random number, etc.

The server replies to the ClientHello message with a ServerHello message, which contains a version number of TLS that is the lower of that suggested by the client and the highest supported by the server, a session ID, a cipher suite selected from the list in the ClientHello message, a random number, etc. If the server should be authenticated, he/she sends his/her certificate to the client. If the server Certificate message does not contain enough data to allow the client to exchange a parameter secret called a pre-master secret, the server sends a ServerKeyExchange message to the client. The server can optionally request a certificate from the client by sending a CertificateRequest message if the server is not anonymous. A ServerHelloDone message is sent by the server to indicate the end of the ServerHello and associated messages.

On receipt of the ServerHelloDone message, the client checks the messages received. The client sends his/her certificate to the server when necessary. A ClientKeyExchange message is always sent by the client after a client Certificate message if it is sent. Otherwise, it is the first message sent by the client after he/she receives a ServerHelloDone message. A pre-master secret, which is a random number generated by the client, is securely exchanged with a

Client		Server
	$\longleftarrow$	HelloRequest*
ClientHello	$\longrightarrow$	
	$\longleftarrow$	ServerHello
	$\longleftarrow$	Certificate*
	$\longleftarrow$	ServerKeyExchange*
	$\longleftarrow$	CertificateRequest*
	$\longleftarrow$	ServerHelloDone
Certificate*	$\longrightarrow$	
ClientKeyExchange	$\longrightarrow$	
CertificateVerify*	$\longrightarrow$	
[ChangeCipherSpec]	$\longrightarrow$	
Finished	$\longrightarrow$	
	$\longleftarrow$	[ChangeCipherSpec]
	$\longleftarrow$	Finished
Application Data	$\longleftrightarrow$	Application Data

Figure 1. The TLS handshake protocol

ClientKeyExchange message. Two methods that is used to exchange a pre-master secret are specified. One uses RSA, and the other Diffie-Hellman. The client can then optionally send a CertificateVerify message to provide explicit verification of the client certificate.

At this moment, the client notifies the server with the change cipher spec protocol that subsequent messages will be protected under the newly negotiated cipher suite and security parameters. Among the security parameters are two symmetric keys that are computed from the two random numbers and the pre-master secret[2]. The two symmetric keys are used to protect messages sent by the client and the server, respectively. The client then sends a Finished message to the server to verify that the handshake process is successful. The Finished message is protected under the just negotiated cipher suite and security parameters. A Finish message is a hash of the security parameters and the handshake messages exchanged so far. The client Finished message is excluded.

On receipt of the client Finished message, the server checks the message. If the message is correct, the server notifies the client with the change cipher spec protocol that subsequent messages will be protected under the newly negotiated cipher suite and security parameters. The server then sends a server Finished message to the client. A server Finish message is a hash of the security parameters and handshake messages exchanged so far. The server Finished message is excluded.

On receipt of the server Finished message, the client checks the message. If the message is correct, the client can exchange application data securely.

[2]Actually the security parameters are computed from the two random numbers, the master secret and some constants. The master secret is computed from the two random numbers, the pre-master secret and a constant.

ClientHello	$A \to B$	Rand_A, ListOfChoices
ServerHello	$B \to A$	Rand_B, SID, Choice
Certificate	$B \to A$	Cert_B
KeyExchange	$A \to B$	$\mathcal{E}_{\mathrm{K}_B}(\mathrm{PMS})$
ClientFinished	$A \to B$	$\mathcal{E}_{\mathrm{ClientKey}}(\mathrm{ClientFinish})$
ServerFinished	$B \to A$	$\mathcal{E}_{\mathrm{ServerKey}}(\mathrm{ServerFinish})$
ClientHello2	$A \to B$	Rand_A, SID
ServerHello2	$B \to A$	Rand_B, SID, Choice
ServerFinished2	$B \to A$	$\mathcal{E}_{\mathrm{ServerKey}}(\mathrm{ServerFinish2})$
ClientFinished2	$A \to B$	$\mathcal{E}_{\mathrm{ClientKey}}(\mathrm{ClientFinish2})$

Figure 2. An abstract handshake protocol

A previously established session and a current session can be resumed and duplicated without exchanging the whole handshake messages. If a client wants to resume or duplicate a session, he/she sets the session ID in a ClientHello message, which is sent to a server. The ClientHello message is protected under the current cipher suite and security parameters. On receipt of the ClientHello message, if the server is willing to resume or duplicate the session, he/she replies with a ServerHello message, which is protected under the current cipher suite and security parameters. The server signals the client to use the newly negotiated security parameters for the session. The same cipher suite is used. A server Finished message is then sent to the client by the server. On receipt of the server Finished message, the client checks the message. If the message is correct, the client signals the server to use the newly negotiated security parameters for the session. A client Finished message is then sent to the server by the client. On receipt of the client Finished message, the server checks the message.

3.2. The Handshake Protocol Analyzed

We show the handshake protocol that has been analyzed in Figure 2. Although the protocol is abstracted away some details, it has the essence of the TLS handshake protocol and is much more complicated than academic protocols such as the NSPK authentication protocol[6]. In the protocol, A denotes a client and B a server. The first six message exchanges are for the full negotiation of a cipher suite and security parameters, and the remaining for the resumption of a previously established session or the duplication of a current session.

Cryptographic primitives used in the protocol are $\mathcal{H}(\cdot)$ (a one-way hash function), $\mathcal{E}_K(\cdot)$ (an encryption function with asymmetric or symmetric key K) and $\mathcal{S}_X(\cdot)$ (a digital signature function with principal X's private key). Basic quantities occurring in the protocol are Rand_X (a random number generated by principal X), ListOfChoices (a list of cipher suites), Choice (a cipher suite), SID (a session ID),

PMS (a pre-master secret) and K_X (principal X's public key). Composite data occurring in the protocol are as follows:

$\mathrm{Cert}_X : X, \mathrm{K}_X, \mathcal{S}_{\mathrm{CA}}(X, \mathrm{K}_X)$
ClientKey : $\mathcal{H}(A, \mathrm{PMS}, \mathrm{Rand}_A, \mathrm{Rand}_B)$
ServerKey : $\mathcal{H}(B, \mathrm{PMS}, \mathrm{Rand}_A, \mathrm{Rand}_B)$
ClientFinish : $\mathcal{H}(\text{"client"}, A, B, \mathrm{SID}, \mathrm{ListOfChoices}, \mathrm{Choice}, \mathrm{Rand}_A, \mathrm{Rand}_B, \mathrm{PMS})$
ServerFinish : $\mathcal{H}(\text{"server"}, A, B, \mathrm{SID}, \mathrm{ListOfChoices}, \mathrm{Choice}, \mathrm{Rand}_A, \mathrm{Rand}_B, \mathrm{PMS})$
ClientFinish2 : $\mathcal{H}(\text{"client"}, A, B, \mathrm{SID}, \mathrm{Choice}, \mathrm{Rand}_A, \mathrm{Rand}_B, \mathrm{PMS})$
ServerFinish2 : $\mathcal{H}(\text{"server"}, A, B, \mathrm{SID}, \mathrm{Choice}, \mathrm{Rand}_A, \mathrm{Rand}_B, \mathrm{PMS})$

Choice is not only a cipher suite, but also you can consider that it includes a version number and a compression algorithm.

We suppose the following: a server always sends his/her certificate to a client when a full handshake process is performed; a server sends neither SeverKeyExchange nor CertificateRequest messages; server Certificate messages also play the role of ServerHelloDone messages; a client sends neither Certificate nor CertificateVerify messages; ChangeCipherSpec messages are implicit; there exists one and only trustable certificate authority denoted by CA; the method used for exchanging pre-master secrets is RSA only; the content of a Finished message is not the hash of the security parameters and handshake messages exchanged, but that of two random numbers, a pre-master secret, etc.; Finished messages exchanged for an abbreviated handshake process are sent in clear.

4. Modeling

4.1. Assumptions

We suppose that there not only exist multiple trustable principals but also multiple malicious (untrustable) principals, and that the cryptosystem used is perfect. Trustable principals exactly follow the protocol, while malicious ones may do something against the protocol as well. The combination and cooperation of malicious principals is modeled as the most general intruder à la Dolev and Yao[4]. The intruder can do the following:

- Eavesdrop any message flowing in the network.
- Glean any quantity from the message; however the intruder can decrypt a ciphertext only if he/she knows the key to decrypt, and cannot compute preimages of a hash unless he/she knows the preimages.
- Fake and send messages based on the gleaned information; however the intruder can encrypt and/or sign something only if he/she knows the key to encrypt and/or sign, and cannot guess unknown secret values.

4.2. Formalization of Messages

Before formalizing messages exchanged in the protocol, we formalize quantities that constitute messages. We declare the following visible sorts and the corresponding data constructors for those quantities:

- `Principal` denotes principals. Two special principals exist; one is the intruder denoted by `intruder` and the other the certificate authority denoted by `ca`. We suppose that `intruder` does not equal `ca`. `Rand` denotes random numbers. `Choice` denotes cipher suites. `Sid` denotes session IDs. `ListOfChoices` denotes lists of cipher suites. Operator `_\in_` is the membership predicate of lists.
- `Secret` denotes secret values that make pre-master secret globally unique and unguessable. `Pms` denotes pre-master secrets. Given two principals a, b and a secret value s, a pre-master secret generated by client a for server b is denoted by $\mathtt{pms}(a, b, s)$.
- `PubKey` denotes public keys. The principal a's public key is denoted by $\mathtt{k}(a)$.
- `Sig` denotes digital signatures of pairs of a principal and a public key. The digital signature of the pair of principal b and public key k signed by principal a is denoted by $\mathtt{sig}(a, b, k)$.
- `Cert` denotes certificates of public keys. Given principal a, public key k and signature g, the certificate that k is a's, which is certified by g, is denoted by $\mathtt{cert}(a, k, g)$.
- `Key` denotes hashes used as symmetric keys to encrypt Finished messages. Given principal a, pre-master secret *pms* and two random numbers $r1, r2$, the hash of those quantities is denoted by $\mathtt{k}(a, pms, r1, r2)$.
- `CFinish` denotes ClientFinish's. Given two principals a, b, session ID i, list l of cipher suites, cipher suite c, two random numbers $r1, r2$ and pre-master secret s, the corresponding ClientFinish is denoted by $\mathtt{cfin}(a, b, i, l, c, r1, r2, s)$.
- `SFinish` denotes ServerFinish's. Given two principals a, b, session ID i, list l of cipher suites, cipher suite c, two random numbers $r1, r2$ and pre-master secret s, the corresponding ServerFinish is denoted by $\mathtt{sfin}(a, b, i, l, c, r1, r2, s)$.
- `CFinish2` denotes ClientFinish2's. Given two principals a, b, session ID i, cipher suite c, two random numbers $r1, r2$ and pre-master secret s, the corresponding ClientFinish is denoted by $\mathtt{cfin2}(a, b, i, c, r1, r2, s)$.
- `SFinish2` denotes ServerFinish2's. Given two principals a, b, session ID i, cipher suite c, two random numbers $r1, r2$ and pre-master secret s, the corresponding ServerFinish2 is denoted by $\mathtt{sfin2}(a, b, i, c, r1, r2, s)$.

• `EncPms` denotes pre-master secrets encrypted by public keys. Pre-master secret *pms* encrypted by public key k is denoted by $\texttt{epms}(k, pms)$.

• `EncCFin` denotes ClientFinish's encrypted by symmetric keys. ClientFinish f encrypted by symmetric key k is denoted by $\texttt{ecfin}(k, f)$.

• `EncSFin` denotes ServerFinish's encrypted by symmetric keys. ServerFinish f encrypted by symmetric key k is denoted by $\texttt{esfin}(k, f)$.

• `EncCFin2` denotes ClientFinish2's encrypted by symmetric keys. ClientFinish2 f encrypted by symmetric key k is denoted by $\texttt{ecfin2}(k, f)$.

• `EncSFin2` denotes ServerFinish2's encrypted by symmetric keys. ServerFinish2 f encrypted by symmetric key k is denoted by $\texttt{esfin2}(k, f)$.

• `Session` denotes quadruples of a cipher suite, two random numbers and a pre-master secret. A quadruple of cipher suite c, two random numbers $r1, r2$ and pre-master secret *pms* is denoted by $\texttt{st}(c, r1, r1, pms)$.

For each data constructor such as `pms`, projection operators such as `client`, `server` and `secret` that return arguments are also defined. For example, $\texttt{client}(\texttt{pms}(a, b, s)) = a$, $\texttt{server}(\texttt{pms}(a, b, s)) = b$ and $\texttt{secret}(\texttt{pms}(a, b, s)) = s$.

Since we suppose that the cryptosystem used is perfect, we can suppose that, given any two different values, the two results of a hash function are different. Therefore, we use the five different visible sorts `Key`, `CFinish`, `SFinish`, `CFinish2` and `SFinish2` for the five kinds of hashes. For a similar reason, we use the four different visible sorts `EncCFin`, `EncSFin`, `EncCFin2` and `EncSFin2` for the four kinds of ciphertexts encrypted by symmetric keys.

We have the 10 operators (data constructors) to denote the 10 kinds of messages. The data constructors are declared as follows:

```
op ch  : Prin Prin Prin Rand ListOfChoices -> Msg
op sh  : Prin Prin Prin Rand Sid Choice    -> Msg
op ct  : Prin Prin Prin Cert               -> Msg
op kx  : Prin Prin Prin EncPms             -> Msg
op cf  : Prin Prin Prin EncCFin            -> Msg
op sf  : Prin Prin Prin EncSFin            -> Msg
op ch2 : Prin Prin Prin Rand Sid           -> Msg
op sh2 : Prin Prin Prin Rand Sid Choice    -> Msg
op cf2 : Prin Prin Prin EncCFin2           -> Msg
op sf2 : Prin Prin Prin EncSFin2           -> Msg
```

`Msg` is the visible sort denoting messages. For data constructor x ($x =$ `ch`, `sh`, `ct`, `kx`, `cf`, `sf`, `ch2`, `sh2`, `sf2`, `cf2`), predicate x`?` is defined, which checks if a given message is x message. Given a term denoting a message, projections `crt`, `src` and `dst` return the first, second and third arguments of the term, respectively, and projections `rand`, `list`, `choice`, `sid`, `cert`, `epms`, `ecfin`, `esfin`, `ecfin2` and `esfin2` return other arguments, respectively, if any.

The first, second and third arguments of each data constructor mean the actual sender (creator), the seeming sender and the receiver of the corresponding message. The first argument is meta-information that is only available to the outside observer and the principal that has sent the corresponding message, and that cannot be forged by the intruder, while the remaining arguments may be forged by the intruder. Therefore suppose that there exists a message in the network. It is true that the principal denoted by the first argument has sent the message. If the first argument is the intruder and the second one is not, the message has been faked by the intruder.

This formalization of messages makes it possible to describe properties such as one that a message received by a principal really originates from the seeming sender of the message. For example, suppose that a principal a receives a message denoted by $\texttt{sf}(b2, b1, a, esfin)$. Since messages are supposed to be never deleted from the network (see the next subsection), if there exists a message denoted by $\texttt{sf}(b1, b1, a, esfin)$ in the network, we can conclude that the message received by a really originates from $b1$.

4.3. Formalization of the Network

The network is modeled as a bag (multiset) of messages, which is used as the storage that the intruder can use. The network is also used as each principal's private memory that reminds the principal to send messages, whose first arguments denote the principal. Any message that has been sent or put once into the network is supposed to be never deleted from the network because the intruder can replay the message repeatedly, although the intruder cannot forge the first argument. Consequently, the emptiness of the network means that no messages have been sent.

The intruder tries to glean seven kinds of quantities from the network as much as possible. The seven kinds of quantities are pre-master secrets, digital signatures and five kinds of ciphertexts.

Random numbers, session IDs, cipher suites, lists of cipher suites and public keys are supposed to be guessable because they are sent in clear and the intruder can glean them without any difficulties. Since symmetric keys, namely ClientKey's and ServerKey's, are never included by any messages, they are never gleaned by the intruder, except those computed from pre-master secrets known by the intruder. Such symmetric keys available to the intruder can be easily computed from pre-master secrets. Therefore, it is not necessary for the intruder to glean symmetric keys. In order that a faked ciphertext used in Finished messages is useful for attacking the protocol, the pre-master secret used to compute the symmetric key must equal that used to compute the hash such as ClientFinish and ServerFinish. Since the pre-master secret used to compute any symmetric key

available to the intruder is also known by the intruder, the pre-master secret must be used to compute a hash such as ClientFinish and ServerFinish so that the hash encrypted by the symmetric key can be useful. Therefore, it is not necessary for the intruder to glean four kinds of hashes used in Finished messages.

The collections of the seven quantities gleaned by the intruder from the network are denoted by the following operators, respectively:

```
op cpms    : Network -> ColPms
op csig    : Network -> ColSig
op cepms   : Network -> ColEncPms
op cecfin  : Network -> ColEncCFin
op cesfin  : Network -> ColEncSFin
op cecfin2 : Network -> ColEncCFin2
op cesfin2 : Network -> ColEncSFin2
```

`Network` is the visible sort denoting networks. `Col`X is the visible sort denoting collections of quantities denoted by visible sort X.

The operators are defined with equations. In this paper, we show the equations that define `cpms`.

```
eq  PMS \in cpms(void) = (client(PMS) = intruder) .
ceq PMS \in cpms(M,NW) = true
    if (kx?(M) and owner(k(epms(M))) = intruder
                and PMS = pms(epms(M))) .
ceq PMS \in cpms(M,NW) = PMS \in cpms(NW)
    if not(kx?(M) and owner(k(epms(M))) = intruder
                  and PMS = pms(epms(M))) .
```

Constant `void` denotes the empty bag. Operator `_\in_` is the membership predicate of collections. Operator `_,_` of `M,NW` is the data constructor of bags. The first equation says that any pre-master secret generated by the intruder is always available to the intruder, and any other pre-master secrets are not at any initial state. Messages from which pre-master secrets can be gleaned are Certificate messages only. If there exits a Certificate message in the network and the ciphertext in the message is encrypted with the intruder's public key, then the pre-master secret in the message can be available to the intruder, which is denoted by the second equation. The third equation says that no pre-master secrets cannot be gleaned from any non-Certificate messages and any Certificate messages whose ciphertexts are not encrypted with the intruder's public key. The remaining operators are defined likewise.

4.4. Formalization of Trustable Principals

Before modeling the behavior of trustable principals, we describe the values observable from the outside of the protocol. We suppose that the network, session states between two principals, the set of used random numbers, the set of used session IDs and the set of used secrets are observable. The observers are denoted by CafeOBJ observation operators `nw`, `ss`, `ur`, `ui` and `us`, respectively, which are declared as follows:

```
bop nw : Protocol -> Network
bop ss : Protocol Prin Prin Sid -> Session
bop ur : Protocol -> URand
bop ui : Protocol -> USid
bop us : Protocol -> USecret
```

`Protocol` is the hidden sort denoting the state space. `URand`, `USid` and `USecret` are the visible sorts denoting sets of random numbers, ones of session IDs and ones of secrets. Let p denote a state of the protocol. `nw`(p) denotes the network, `ur`(p) the set of used random numbers, `ui`(p) that of used session IDs and `us`(p) that of used secrets in the state. Besides, let a, b denote principals and i a session ID. `ss`(p, a, b, i) denotes the principal a's session state identified by session ID i with principal b in the state.

The behavior of trustable principals is modeled by 12 kinds of transitions. 10 of them correspond to sending the 10 kinds of messages. The remaining two correspond to clients' receiving ServerFinished messages and servers' receiving ClientFinished2 messages, respectively. The 12 kinds of transitions are denoted by CafeOBJ action operators `chello`, `shello`, `cert`, `kexch`, `cfin`, `sfin`, `compl`, `chello2`, `shello2`, `sfin2`, `cfin2` and `compl2`. The operators are declared as follows:

```
bop chello  : Protocol Prin Prin Rand
                                ListOfChoices -> Protocol
bop shello  : Protocol Prin Rand Sid Choice
                                          Msg -> Protocol
bop cert    : Protocol Prin Msg Msg         -> Protocol
bop kexch   : Protocol Prin Secret Msg Msg
                                          Msg  -> Protocol
bop cfin    : Protocol Prin Secret Msg Msg
                                      Msg Msg -> Protocol
bop sfin    : Protocol Prin Msg Msg Msg Msg
                                          Msg -> Protocol
bop compl   : Protocol Prin Secret Msg Msg
                                 Msg Msg Msg Msg -> Protocol
bop chello2 : Protocol Prin Prin Secret Rand
                                          Sid -> Protocol
bop shello2 : Protocol Prin Rand Msg        -> Protocol
bop sfin2   : Protocol Prin Msg Msg         -> Protocol
bop cfin2   : Protocol Prin Secret Msg Msg
                                          Msg -> Protocol
bop compl2  : Protocol Prin Msg Msg Msg Msg -> Protocol
```

The 12 action operators are defined with equations. In this paper, we show the equations for `cert`, which are declared as follows:

```
op c-cert : Protocol Prin Msg Msg -> Bool
eq c-cert(P,B,M1,M2) = (M1 \in nw(P) and M2 \in nw(P)
     and ch?(M1) and sh?(M2) and dst(M1) = B and
     crt(M2) = B and src(M2) = B and src(M1) = dst(M2)
     and choice(M2) \in list(M1)) .

ceq nw(cert(P,B,M1,M2))
    = ct(B,B,dst(M2),cert(B,k(B),sig(ca,B,k(B))))
       , nw(P) if c-cert(P,B,M1,M2) .
eq  ss(cert(P,B,M1,M2),A2,B2,I2) = ss(P,A2,B2,I2) .
eq  ur(cert(P,B,M1,M2))          = ur(P) .
eq  ui(cert(P,B,M1,M2))          = ui(P) .
eq  us(cert(P,B,M1,M2))          = us(P) .
ceq cert(P,B,M1,M2) = P if not c-cert(P,B,M1,M2) .
```

That principal a has sent message m_1 to principal b is denoted by that m_1 exists in the network, both `crt`(m_1) and

$\texttt{src}(m_1)$ equal a and $\texttt{dst}(m_1)$ equals b. In order for principal a to receive message m_2 from principal b, there must exist m_2 in the network such that $\texttt{src}(m_2)$ equals b and $\texttt{dst}(m_2)$ equals a. But, $\texttt{crt}(m_2)$ may not equal b but the intruder. The other action operators are defined likewise.

4.5. Formalization of the Intruder

Part of the intruder has been modeled as the network. We have defined what information the intruder can glean from the network. We next describe what messages the intruder fakes based on the gleaned information.

We have 15 kinds of transitions that denote the intruder's faking messages, which are denoted by 15 CafeOBJ action operators. The effective condition of any transition corresponding to each of the action operators is that the necessary information is available to the intruder.

In this paper, we show the two action operators for faking ServerFinished messages, which are declared as follows:

```
bop fakeSfin1 : Protocol Prin Prin EncSFin -> Protocol
bop fakeSfin2 : Protocol Prin Prin Sid ListOfChoices
                      Choice Rand Rand Pms -> Protocol
```

The 15 action operators are defined with equations. In this paper, we show the equations for `fakeSfin2`, which are declared as follows:

```
op c-fakeSfin2 : Protocol Prin Prin Sid ListOfChoices
                         Choice Rand Rand Pms -> Bool
eq c-fakeSfin2(P,B,A,I,L,C,R1,R2,PMS)
   = PMS \in cpms(nw(P)) .

ceq nw(fakeSfin2(P,B,A,I,L,C,R1,R2,PMS))
    = sf(intruder,B,A,esfin(k(B,PMS,R1,R2),
         sfin(A,B,I,L,C,R1,R2,PMS))) , nw(P)
    if c-fakeSfin2(P,B,A,I,L,C,R1,R2,PMS) .
eq  ss(fakeSfin2(P,B,A,I,L,C,R1,R2,PMS),B2,A2,I2)
    = ss(P,B2,A2,I2) .
eq  ur(fakeSfin2(P,B,A,I,L,C,R1,R2,PMS)) = ur(P) .
eq  ui(fakeSfin2(P,B,A,I,L,C,R1,R2,PMS)) = ui(P) .
eq  us(fakeSfin2(P,B,A,I,L,C,R1,R2,PMS)) = us(P) .
ceq fakeSfin2(P,B,A,I,L,C,R1,R2,PMS)
    = P if not c-fakeSfin2(P,B,A,I,L,C,R1,R2,PMS) .
```

The other action operators are defined likewise.

5. Analysis

5.1. Verified Properties

The informal descriptions of the verified properties for the protocol are first given.

1. Pre-master secrets cannot be leaked.

2. If a trustable client receives a ServerFinished message that conforms to the protocol and seems to have been sent by a server, then the message really originates from the server.

3. If a trustable client receives a ServerFinished2 message that conforms to the protocol and seems to have been sent by a server, then the message really originates from the server.

4. If a trustable client receives a ServerHello message, a Certificate message and a ServerFinished message that conform to the protocol and seem to have been sent by a server, then the ServerHello and Certificate messages really originate from the server.

5. If a trustable client receives a ServerHello2 message and a ServerFinished2 message that conform to the protocol and seem to have been sent by a server, then the ServerHello2 message really originate2 from the server.

Pre-master secrets are the most fundamental parameters in the protocol because all security parameters are computed based on them. The first property assures the users of the protocol that secret security parameters are really secret. The second property guarantees that when a client has negotiated a cipher suite and security parameters with the corresponding server by a full handshake process, the server has really agreed on them. The third property guarantees that when a client has negotiated a cipher suite and security parameters with the corresponding server by an abbreviated handshake process, the server has really agreed on them. The fourth property guarantees that when a client has negotiated a cipher suite and security parameters with the corresponding server by a full handshake process, the cipher suite and some of the security parameters have been suggested by the server. The fifth property guarantees that when a client has negotiated a cipher suite and security parameters with the corresponding server by an abbreviated handshake process, the cipher suite and some of the security parameters have been suggested by the server.

The properties are formalized as CafeOBJ terms in order to be formally verified. In this paper, we show the definitions of the first and second properties.

The CafeOBJ term denoting the first property is as follows:

```
op inv1 : Protocol Pms -> Bool
eq inv1(P,PMS) = (PMS \in cpms(nw(P)) implies
   (client(PMS) = intruder or server(PMS) = intruder)) .
```

The term says that if a pre-master secret is available to the intruder, the pre-master secret has been generated by the intruder or a client has generated it for a session with the intruder. This implies that the intruder cannot obtain any pre-master secrets for sessions in which the intruder is not involved.

The CafeOBJ term denoting the second property is as follows:

```
op inv2 : Protocol Prin Prin Prin Rand Rand
                ListOfChoices Choice Sid Secret -> Bool
eq inv2(P,A,B,B1,R1,R2,L,C,I,S)
   = (not(A = intruder) and
      sf(B1,B,A,esfin(k(B,pms(A,B,S),R1,R2),
         sfin(A,B,I,L,C,R1,R2,pms(A,B,S)))) \in nw(P)
      implies
```

```
    sf(B,B,A,esfin(k(B,pms(A,B,S),R1,R2),
       sfin(A,B,I,L,C,R1,R2,pms(A,B,S)))) \in nw(P)) .
```

The term says that if a trustable client, which means that the client is not the intruder, receives a ServerFinished message that seems to have been sent by a server, which means that the actual sender denoted by `B1` may be the intruder, conforms to the protocol, and uses a pre-master secret generated by the client for the server, then the message really originates from the server.

We need 13 more properties to prove the five properties. Five of the properties, including the fourth and fifth ones, have been proved by case analyses with other properties, and the remaining ones by induction on the number of transitions applied.

5.2. Verification

We describe part of the proof score of `inv2`. We declare the operator denoting the basic formula to prove in each inductive case as follows:

```
op istep2 : Prin Prin Prin Rand Rand ListOfChoices
                          Choice Sid Secret -> Bool
eq istep2(A,B,B1,R1,R2,L,C,I,S)
   = inv2(p,A,B,B1,R1,R2,L,C,I,S)
     implies inv2(p',A,B,B1,R1,R2,L,C,I,S) .
```

`p` and `p'` are constants of hidden sort `Protocol`. `p` denotes an arbitrary state, and `p'` a successor state of `p`.

Let `sfin1`, `sfin2` and `sfin3` be the following terms, respectively:

```
sf(intruder,b10,a10,esfin(k(b10,pms10,r10,r20),
          sfin(a10,b10,i10,l10,c10,r10,r20,pms10)))

sf(b1,b,a,esfin(k(b,pms(a,b,s),r1,r2),
          sfin(a,b,i,l,c,r1,r2,pms(a,b,s))))

sf(b,b,a,esfin(k(b,pms(a,b,s),r1,r2),
          sfin(a,b,i,l,c,r1,r2,pms(a,b,s))))
```

where all the constants except for `intruder` denote arbitrary values of the intended sorts. For example, `r1`, `r2`, `r10` and `r20` denote arbitrary values of visible sort `Rand`.

Let us consider the inductive case that action operator `fakeSfin2` preserves `inv2`. When the effective condition of `fakeSfin2` is true, the case is split into the following five sub-cases:

1. $(\texttt{sfin1} \neq \texttt{sfin2}) \wedge (\texttt{sfin1} \neq \texttt{sfin3})$
2. $(\texttt{sfin1} \neq \texttt{sfin2}) \wedge (\texttt{sfin1} = \texttt{sfin3})$
3. $(\texttt{sfin1} = \texttt{sfin2}) \wedge (\texttt{b} = \texttt{intruder})$
4. $(\texttt{sfin1} = \texttt{sfin2}) \wedge (\texttt{b} \neq \texttt{intruder}) \wedge (\texttt{a} = \texttt{intruder})$
5. $(\texttt{sfin1} = \texttt{sfin2}) \wedge (\texttt{b} \neq \texttt{intruder}) \wedge (\texttt{a} \neq \texttt{intruder})$

The first 4 sub-cases do not need any other predicates to strengthen the inductive hypothesis, but the fifth sub-case needs `inv1` to strengthen the inductive hypothesis.

We show the proof passage corresponding to the fifth sub-case.

```
open ISTEP
-- arbitrary objects
 ops a10 b10 : -> Prin .      op i10 : -> Sid .
 op l10 : -> ListOfChoices .  op c10 : -> Choice .
 ops r10 r20 : -> Rand .      op pms10 : -> Pms .
-- assumptions
 eq pms(a,b,s) \in cpms(nw(p)) = true .
 --
 eq b1 = intruder .  eq r10 = r1 .  eq i10 = i .
 eq l10 = l .  eq c10 = c .  eq pms10 = pms(a,b,s) .
 eq r20 = r2 .  eq a10 = a .  eq b10 = b .
 --
 eq (b = intruder) = false . eq (a = intruder) = false .
-- successor state
 eq p' = fakeSfin2(p,b10,a10,
                   i10,l10,c10,r10,r20,pms10) .
-- check if the predicate is true.
 red inv1(p,pms(a,b,s))
     implies istep2(a,b,b1,r1,r2,l,c,i,s) .
close
```

Constants such as `r1` and `r2` that are not declared in this proof passage are declared in a module, say `INV`, imported by `ISTEP`. The first equation means that the effective condition is true, and the second through 10th equations mean that `sfin1` equals `sfin2`. The equation $\texttt{sfin1} = \texttt{sfin2}$ can be deduced from the 9 equations by rewriting, but the nine equations cannot be deduced from the one equation by rewriting. That is why the nine equations are used. In this passage, `inv1(p,pms(a,b,s))` is used to strengthen the inductive hypothesis `inv2(p,a,b,b1,r1,r2,l,c,i,s)`.

5.3. Other Results

We attempted to verify the servers' counterparts of the second and third properties, which are informally described as follows:

$2'$. If a trustable server receives a ClientFinished message that conforms to the protocol and seems to have been sent by a client, then the message really originates from the client.

$3'$. If a trustable server receives a ClientFinished2 message that conforms to the protocol and seems to have been sent by a client, then the message really originates from the client.

The verifications did not succeeded, however, and we found that the properties do not hold for the protocol.

There is a counterexample of property $2'$, which is shown as follows:

(1) $\texttt{ch}(a', a, b, r_{a'}, l)$
(2) $\texttt{sh}(b, b, a, r_b, sid, c)$
(3) $\texttt{ct}(b, b, a, \texttt{cert}(b, k_b, \texttt{sig}(\texttt{ca}, b, k_b)))$
(4) $\texttt{kx}(a', a, b, \texttt{epms}(k_b, pms'))$
(5) $\texttt{cf}(a', a, b, \texttt{ecfin}(k(a, pms', r_{a'}, r_b), \texttt{cfin}(a, b, sid, l, c, r_{a'}, r_b, pms')))$

This shows that even if server b receives a ClientFinished message that the server expects to receive from client a, the message actually originates from client a'.

Suppose that there exists a session between server b and client a' but b believes that he/she has established the session with client a. Such a session can be established as shown in the just mentioned example. There is then a counterexample of property $3'$, which is shown as follows:

(1) $\mathtt{ch2}(a', a, b, r_{a'}, sid)$
(2) $\mathtt{sh}(b, b, a, r_b, sid, c)$
(3) $\mathtt{sf}(b, b, a, \mathtt{esfin2}(k(b, pms', r_{a'}, r_b),$
$\mathtt{sfin2}(a, b, sid, c, r_{a'}, r_b, pms')))$
(4) $\mathtt{cf}(a', a, b, \mathtt{ecfin2}(k(a, pms', r_{a'}, r_b),$
$\mathtt{cfin2}(a, b, sid, c, r_{a'}, r_b, pms')))$

Since clients are not authenticated in the protocol that has been analyzed, it might be obvious that properties $2'$ and $3'$ do not hold in the protocol. While we were reading the specification of TLS, however, we did not notice that the two properties do not hold in TLS where clients are not authenticated. We believe that formal analyses of security protocols also help engineers notice what they might not notice just by reading informally written specifications of the protocols. The two counterexamples demonstrate this.

The two examples also guarantee that if clients use TLS where they are not authenticated, they cannot be identified and they are anonymous.

In the protocol shown in Figure 2, a ServerFinished2 message precedes a ClientFinished2 message. We have also verified that the five properties described in the previous subsection hold in the protocol where a ClientFinished2 message precedes a ServerFinished2 message. When a specification is slightly changed, our method, namely the OTS/CafeOBJ method, makes it possible to easily adjust proof scores.

6. Related Work

Mitchell, et al.[7] use the model checker Murφ[3] to check seven simple protocols derived from the SSL 3.0 handshake protocol. The primal reason why they have analyzed the protocols is to identify the purpose of certain message fields (version number, nonce, etc.) in some steps of the protocol. The analysis starts with the simplest version of the handshake protocol from which some fields are omitted, and the fields are gradually added to the protocol. The first six protocols have been found badly flawed and the model checker has found many attacks. The model checker has been used to check the final protocol with two clients, one server, no more than two simultaneous open sessions per server and no more than one resumption per session, and no attacks have been discovered.

Paulson[11] analyzes the TLS handshake protocol with his inductive method[10] that is supported by the proof assistant Isabelle/HOL[8]. In the protocol analyzed by Paulson, servers always sent their certificates to clients, the key exchange method considered is RSA, and clients optionally send their certificates and ClientKeyExchange messages to servers. In his model of the protocol, a malicious principal called the spy, which corresponds to the intruder in our model, is taken into account and it is supposed that any session key, if used, may end up in the hands of the spy, which is denoted by the rule *Oops*. One of the results of the analysis is that session resumption turns out to be safe even if the spy has obtained session keys from earlier sessions.

7. Conclusion

It took a couple of weeks to read the TLS specification and obtain the abstract handshake protocol, and it took about one week to complete the analysis. Among the reasons why the analysis was completed in such period are that (1) most part of a proof score that an OTS has a property can be reused for other proof scores that the OTS has other properties and (2) the method used allows us to write proofs even if we do not know how to construct the proofs in very detail, namely what equations (or deductive rules) should be applied to terms denoting formulas to prove.

References

[1] R. Diaconescu and K. Futatsugi. *CafeOBJ report*. AMAST Series in Computing, 6. World Scientific, Singapore, 1998.

[2] T. Dierks and C. Allen. The TLS protocol version 1.0. Request for Commnets: 2246, `http://www.ietf.org/rfc/rfc2246.txt`, 1999.

[3] D. L. Dill. The Murφ verification system. In *CAV '96*, LNCS 1102, pages 390–393. Springer, 1996.

[4] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Trans. Inform. Theory*, IT-29:198–208, 1983.

[5] J. Hsiang and N. Dershowitz. Rewrite methods for clausal and nonclausal theorem proving. In *ICALP '83*, volume 154 of *LNCS*, pages 331–346. Springer, 1983.

[6] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS '96*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.

[7] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0 and related protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[8] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, Berlin, 2002.

[9] K. Ogata and K. Futatsugi. Proof scores in the OTS/CafeOBJ method. In *FMOODS 2003*, volume 2884 of *LNCS*, pages 170–184. Springer, 2003.

[10] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[11] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Trans. Infom. and Sys. Sec.*, 2(3):332–351, 1999.