TESLA Source Authentication Protocol Verification Experiment in the Timed OTS/CafeOBJ Method: Experiences and Lessons Learned*

Iakovos OURANOS††, Nonmember, Kazuhiro OGATA††, Member, and Petros STEFANEAS†††, Nonmember

SUMMARY In this paper we report on experiences gained and lessons learned by the use of the Timed OTS/CafeOBJ method in the formal verification of TESLA source authentication protocol. These experiences can be a useful guide for the users of the OTS/CafeOBJ, especially when dealing with such complex systems and protocols.

key words: source authentication, TESLA protocol, CafeOBJ, verification, timed OTS, lessons learned

1. Introduction

The Timed OTS/CafeOBJ method [1], is a version of the OTS/CafeOBJ [2] method for modeling real-time systems. The main advantage of these methods is that system’s specification and verification is written in terms of equations, which are the most fundamental logical formulas, easier to learn and use than other formal methods. Although the OTS/CafeOBJ method has been used in several complex, real life cases [3]–[5], the real time version of it has been used only for simple systems [1].

In this paper we apply the Timed OTS/CafeOBJ method to the modeling and verification of basic TESLA protocol [6], [7], the simpler but yet very sophisticated version of TESLA protocol. TESLA, which stands for Time Efficient Source Loss-Tolerant Authentication Protocol, is a source authentication protocol used in multicast settings. It achieves properties of asymmetric cryptography by using symmetric primitives and time synchronization. Authentication of a data packet is based on information of the next previous packets. The protocol finds application to the continuous authentication of radio and TV Internet broadcasts, authenticated data distribution by satellite, and has been published as an IETF standard [8].

In the OTS/CafeOBJ method, a protocol, algorithm, or software system is modeled as an Observational Transition System (OTS), which is a kind of transition system that can be written straightforwardly in terms of equations. Next, the OTS is described in CafeOBJ algebraic specification language [9]. Properties to verify are then expressed as CafeOBJ terms, and proof scores showing that the specified OTS model has desired properties are also written in CafeOBJ. Finally, proof scores are executed with the CafeOBJ system.

When dealing with real time systems, the system specification is extended with special data types called clock observers that model timing issues, and a special time advancing transition, and OTSs are evolved to Timed OTSs. This approach can be seen as an application of the old-fashioned recipe of Abadi and Lamport [10].

The purpose of this paper is to provide information to newcomers and experienced users of the OTS/CafeOBJ method, about the lessons learned from the “TESLA verification experiment” which is an ideal representative of a complex protocol case study. Such complex protocols require an in depth study before specifying them which is not straightforward in a first attempt, as shown in our experience gained. The process of understanding continues during the formal description of the protocol/system, and in general many revisions of the specification are required, even when verification has started. Since fully understanding a complex system is impossible at an initial stage, specification writing can start when reaching an adequate level of system and property understanding. Keep in mind that the process of understanding is continuous and interactive. You may want to use diagrammatic systems specification animation by hand, symbolic systems specification animation by CafeOBJ and verification attempts by writing proof scores in CafeOBJ to get better understanding of systems and properties.

The rest of the paper is organized as follows: Section 2 presents the TESLA verification experiment, providing the required information to the readers about the Timed OTS/CafeOBJ method, the TESLA protocol and the formal analysis of it. Section 3, which is the main part of the paper, discusses the experiences gained and the lessons learned. Section 4 presents related works, while Sect.5 concludes the paper and mentions future issues.
2. TESLA Verification Experiment

2.1 Timed Observational Transition Systems

$U$ is the universal state space (the set of all possible states) and $R^+ \cup \{0\}$ is the set of non-negative real numbers. Sets and types may be interchangeably used. $\text{Bool}$ is the type for truth values.

**Definition 1 (TOTS).** A TOTS $S$ consists of $<O, I, T \cup \{\text{tick}\}>$ such that

- $O$: A set of observers. Each observer is a function $o: UD_1 \ldots UD_m \rightarrow D_o$. If $D_o$ is a subset of $R^+ \cup \{0\}$, $o$ is called a clock observer. Otherwise, $o$ is called a discrete observer. The equivalence between two states $u_1, u_2$ (denoted as $u_1 = u_2$) is defined wrt valuations returned by the observers. Among clocks is now: $U \rightarrow R^+$ that plays a master clock and initially returns 0.
- $I$: The set of initial states such that $I \subseteq U$.
- $T \cup \{\text{tick}\}$: A set of transitions. Each transition is a function $t : UD_1 \ldots UD_m \rightarrow U$. Each transition $t$, together with any other parameters $y_1, \ldots, y_n$, preserves the equivalence between two states. This means that for all states $s_1, s_2$ such that $s_1$ is equivalent to $s_2$ and for each transition, let $s_1', s_2'$ be the successors of $s_1, s_2$ with respect to the transition, and then $s_1'$ is equivalent to $s_2'$.

Each $t$ has the effective condition that consists of the non-timing part $ct$ and the timing part $tc-t$ whose types are $UD_1 \ldots UD_m \rightarrow \text{Bool}$. If $ct(u_1, y_1, \ldots, y_n) \land tc-t(u_1, y_1, \ldots, y_n)$ does not hold, then $t(u_1, y_1, \ldots, y_n) = s$. $\text{tick}$ is a time advancing transition whose type is $UR^+ \rightarrow U$. If $c-\text{tick}(u, r)$ holds, now($\text{tick}(u, r)$) is now($u$)+$r$, namely advancing the master clock by $r$. Any application of $\text{tick}$ does not affect the values returned by any observers except for now, and the value returned by now is only affected by applications of $\text{tick}$.

For each $t \in T$, there are two clocks $l_t : UD_1 \ldots UD_m \rightarrow R^+$ and $u_t : UD_1 \ldots UD_m \rightarrow (R^+ \cup \{0\})[\text{max}]$. The two clocks return the lower and upper bounds of $t$, and are used to force $t$ to be applied during the interval. $tc-t(u, y_1, \ldots, y_n)$ is $l(u, y_1, \ldots, y_n) \leq now(u)$.

For each $t \in T$, there are two functions $d_{\min}$ and $d_{\max}$ whose types are the same as $l_t$ and $u_t$. $d_{\min}$ and $d_{\max}$ give the minimum and maximum delays of $t$, which are used to calculate the values returned by $l_t$ and $u_t$ as follows:

- Let $\text{init}$ be an arbitrary initial state.

$$
n(t(\text{init}, y_1, \ldots, y_n)) = \begin{cases} 
   d_{\min}(\text{init}, y_1, \ldots, y_n) & c - t(\text{init}, y_1, \ldots, y_n) = \text{false} \\
   0 & \text{otherwise}
\end{cases}
$$

$$
n(t(\text{init}, y_1, \ldots, y_n)) = \begin{cases} 
   d_{\max}(\text{init}, y_1, \ldots, y_n) & c - t(\text{init}, y_1, \ldots, y_n) = \text{true} \\
   0 & \text{otherwise}
\end{cases}
$$

- Assume that $c \cdot t(u_1, z_1, \ldots, z_m) \land lt(u_1, z_1, \ldots, z_m) \leq now(u)$.

  Let $u'$ be $t'(u_1, z_1, \ldots, z_m)$ and $t$ be any other transition than $t'$.

$$
l'(u', y_1, \ldots, y_n) = \begin{cases} 
   d_{\min}(u_1, y_1, \ldots, y_n) & c - t'(u', y_1, \ldots, y_n) = \text{false} \\
   0 & \text{otherwise}
\end{cases}
$$

**Definition 2 (Execution).** An execution of $S$ is an infinite sequence $u_0, u_1, \ldots$ of states satisfying

- Initiation: $u_0 \in I$.
- Consecution: For each natural number $i$, there exists $t \in T$ such that $u_{i+1} = S t(u_i, y_1, \ldots, y_n)$ for some parameters $y_1, \ldots, y_n$ or $u_{i+1} = \text{tick}(u_i, r)$ for some $r$.
- Time Divergence: As $i$ increases, now($u_i$) increases without bound.

Let $E_S$ be the set of all executions obtained from $S$.

**Definition 3 (Reachable State).** A state $u$ is called reachable wrt $S$ iff there exists an execution $e \in E_S$ such that $u \in e$. Let $R_S$ be the set of all reachable states wrt $S$.

**Definition 4 (Invariant).** A predicate $p: U \rightarrow \text{Bool}$ is called invariant wrt $S$ iff $p$ holds in all reachable states, namely $(\forall u : R_S) p(u)$.

2.2 Specifying and Verifying TOTS in CafeOBJ

A TOTS is specified in CafeOBJ as an OTS. For specifying TOTSs in CafeOBJ, however, we prepare one module called TIMEVAL where extended non-negative real numbers are specified. TIMEVAL is declared with mod*. The signature of the module is as follows:

Zero, NzReal+, <Real+> [NzReal+ + Inf <NzTimeval> [Real+ + NzTimeval <Timeval> op \theta : ->Zero op \infty : ->Inf op <= : Timeval Timeval ->Bool op <=* : Timeval Timeval ->BooI op += : Timeval Timeval ->Timeval [assoc comm] op =>+ : Real+ + Real+ ->Real+ [assoc comm] op <=* : Timeval Timeval ->BooI [comm]

Zero, NzReal, Real+, Inf, NzTimeval and Timeval are visible sorts denoting $[0], R^+ \cup \{0\}, (R^+ \cup \{0\}) \cup \{\infty\}$ and $R^+ \cup \{\infty\}$. Constants $\theta$ and $\infty$ denote $\theta$ and $\infty$. The operator $+.+$ adds two extended non-negative real numbers, the operator $<=$ checks if one extended non-negative real number is less than the other, the operator $<=$ checks if one extended non-negative real number is less than or equal to the other and the operator $<=$ checks if two extended non-negative real numbers are equal. The properties of the operators are specified in equations. Among equations are:

\[eq X \times 0 = false\]
eq \( X < \infty = true \).
\[
\begin{align*}
\text{ceq } X + T1 < X + T2 &= true \text{ if } T1 < T2 . \\
\text{ceq } T < T1 + T2 &= true \text{ if } T < T2 .
\end{align*}
\]

where \( X \) is CafeOBJ variable of the sort \( \text{Real+} \) and \( T, T1, \) and \( T2 \) are variables whose sort is \( \text{Timeval} \).

The same techniques used to verify that an OTS enjoys invariant properties, namely writing proof scores in CafeOBJ, can be used to verify that a TOTS enjoys invariant properties.

2.3 TESLA Protocol

Timed Efficient Stream Loss Tolerant Authentication (TESLA) protocol is a protocol used in broadcast settings for source authentication. It achieves properties of asymmetric cryptography by using symmetric primitives (except for the first digitally signed packet) and time synchronization. Authentication of a packet is based on information of the next and previous packets.

Basic TESLA, which is the simpler but sophisticated version of the protocol and applies the basic ideas in a one-to-one setting, informally works as follows: An initial authentication is achieved using a public key signature. The subsequent messages are authenticated using Message Authentication Codes (MACs) linked back to the initial signature.

In message \( n-1 \), the sender \( S \) generates a key \( k_n \) and transmits \( f(k_n) \) to the receiver \( R \), as a commitment to that key, where \( f \) is a suitable cryptographic hash function.

In message \( n \), \( S \) sends a data packet \( m_n \), authenticated using a MAC with key \( k_n \). The key is revealed in message \( n+1 \).

Each receiver checks that the received key \( k_n \) corresponds to the commitment received in message \( n-1 \), verifies the MAC in message \( n \), and then accepts the data packet \( m_n \) as authentic. Message \( n \) also contains a commitment to the next key \( k_{n+1} \), authenticated by the MAC, thus allowing a chain of authentications. The messages exchanged in Basic TESLA are as follows:

\[\text{Init Message: } R -> S: n_R \]
\[\text{Reply Message: } S -> R: f(k_1), n_R, \{f(k_1), n_R\}_{PK(S)} \]
\[\text{Msg}_1: S -> R: d_1, f(k_2), MAC(k_1, d_1, f(k_2)) \]
\[\text{Msg}_{n-1}: S -> R: d_n, f(k_{n-1}), MAC(k_{n-1}, d_n, f(k_{n-1}), k_{n-1}), n > 1. \]

where \( n_R \) is a nonce generated by the receiver to ensure freshness and \( d_1, d_n \) the data transmitted.

The protocol requires an important time synchronization assumption, the security condition: the receiver will not accept message \( n \) if it arrives after the sender might have sent message \( n+1 \), otherwise an intruder can capture message \( n+1 \), and use the key \( k_n \) from within it to fake a message \( n \).

2.4 Formal Analysis of TESLA Protocol

2.4.1 Outline of Data Types Specification

We sketch the basic assumptions of protocol’s specification and give a brief explanation of data types used. For more details on specification of the protocol someone can consult [26].

We suppose that there exist untrustable nodes as well as trustable ones. Trustable nodes exactly follow the protocol, but untrustable ones may do something against the protocol as well, namely eavesdropping and/or faking of messages. The combination and cooperation of untrustable nodes is modelled as the most general intruder [11]. The cryptosystem used is perfect and so, the intruder can do the following:

- Eavesdrop any message flowing in the network.
- Glean any nonces, data, commitments, keys, message authentication codes (MACs) and signatures from the message; however the intruder can decrypt an encrypted text only if he knows the corresponding key to decrypt.
- Fake and send messages based on the gleaned information; however the intruder cannot guess unknown data.

We first formalize data types that constitute messages in terms of order-sorted algebras. We declare the visible sorts and the corresponding data constructors for those data types. For example, \( \text{Sender} \) models the set of agents that participate in the protocol as server. Two special sender nodes are \( \text{enemy} \) denoting a malicious intruder, and \( \text{server} \) modeling the legitimate server, which are declared as constants of the sort \( \text{Sender} \). Other data types used are: \( \text{Receiver} \) which denotes the set of receivers of the protocol, \( \text{Data} \) which models data to be sent by the sender, \( \text{Key} \) that denotes the symmetric key used for the formation of commitments and MACs (we assume that is used the same key), etc.

Having declared the above data types we then formalize the four different kind of messages exchanged in the protocol:

For example, the initial message \( (\text{im}) \) that a receiver (client) sends to the sender (server) to initiate a session, contains a nonce to ensure freshness, in clear. This is the only message sent by the receiver agent. The constructor of the message is

\[\text{op im : Receiver Receiver Sender Nonce -> Msg} \]

The first argument is meta-information that is only available to the outside observer and the node that has sent the corresponding message, and cannot be forged by the intruder, while the remaining arguments may be forged by the intruder. So, if the first argument is the \( \text{enemy} \) and second one is not, then the message has been faked by the intruder. Second and third arguments are the seeming sender and receiver, while the last argument is the nonce created by
the sender of the message (i.e. the Receiver) for the server (i.e. the Sender), using a fresh random number. Projections \texttt{crt-im}, \texttt{src-im}, \texttt{dst-im} return the first (actual creator), second (seeming sender), and third (receiver) arguments of each message. A predicate \texttt{im?}, checks if the given message is of the type \texttt{im}. Finally, \texttt{j} returns the identification number of the message, which is zero (\texttt{0}) for message \texttt{im}. Similarly are declared reply message (\texttt{rm}), (\texttt{m1}) that models the first message that sends some data, and the \texttt{n-th} message (\texttt{mn}) which contains the data, the commitment to the key used in the next message \texttt{(kn+1)}, the key used in the previous message \texttt{(kn-1)} and all of them encrypted with the \texttt{k}n in a message authentication code. We also add the index of the message, \texttt{n}.

The network is then modeled as a multiset of messages, which is used as the storage that the intruder can use. Any message that has been sent or put into the network is supposed to be never deleted from the network. As a consequence, the emptiness of the network means that no messages have been sent.

The intruder tries to glean seven kinds of quantities from the network. These are the nonces, data, commitments, the keys, two kinds of the message authentication codes and the signatures. The collections of these quantities are denoted by corresponding operators. For the case of nonces we have:

\texttt{op nonces : Network \rightarrow ColNonces .}

\texttt{Network} is the visible sort denoting networks. \texttt{ColNonces} is the visible sort denoting collections of nonces. Given a snapshot \texttt{nw} of the network, \texttt{nonces(nw)} denotes the collection of nonces appeared in the \texttt{m1} message available to the intruder. The equations for nonces are as follows:

\begin{align*}
\texttt{eq N \ \backslash in \ nonces(void) = (creator(N) = enemy) .}
\texttt{ceq N \ \backslash in \ nonces(M,NW) = true \ if \ im?(M) \ and \ n(M) = N .}
\texttt{ceq N \ \backslash in \ nonces(M,NW) = true \ if \ rm?(M) \ and \ n(c(M)) = N \ and}
\texttt{p(pk(c(M))) = enemy .}
\texttt{ceq N \ \backslash in \ nonces(M,NW) = true \ if \ rm?(M) \ and \ n(M) = N .}
\texttt{ceq N \ \backslash in \ nonces(M,NW) = \ \texttt{N \ \backslash in \ nonces(NW)\ if}}
\texttt{not (im?(M) \ and \ n(M) = N) \ and}
\texttt{not (rm?(M) \ and \ n(c(M)) = N \ and}
\texttt{p(pk(c(M))) = enemy) \ and \ not (rm?(M) \ and \ n(M) = N) .}
\end{align*}

\texttt{Constant void} denotes the empty bag, while \texttt{N}, \texttt{M}, \texttt{NW} are \texttt{CafeOBJ} variables for \texttt{Nonce}, \texttt{Msg} and \texttt{Network}, respectively. Operator \texttt{\backslash in} is the membership predicate of collection, while \texttt{\_} is the data constructor of bags. So, \texttt{M,NW} denotes the network obtained by adding message \texttt{M} to the network \texttt{NW}. The first equation says that initially, the intruder’s nonce is the only available to him. The second equation says that if there exists a message \texttt{M} of the type \texttt{im} in the network, then the nonce \texttt{N} of the message is available to the intruder. In the case of an \texttt{rm} message, we have two subcases: The nonce sent in clear is available to the intruder (Eq. (4)), while the nonce encrypted with sender’s private key in the signature is available to the intruder only if the key belongs to the intruder (Eq. (4)). These are the only nonces available to the intruder.

2.4.2 Specifying the Behavior of the Protocol as a TOTS

Having specified the data part of the specification, we proceed to the specification of the behavior of the protocol in the module \texttt{TESLA}, as an Observational Transition System with real time extensions. The assumptions made and some operators are as follows:

1. Time constraints for sending and receiving messages \texttt{m1} and \texttt{mn}.
2. Ordering of packets using an integer packet \texttt{id}. We assume that messages \texttt{im} and \texttt{rm} have \texttt{id} = 0, \texttt{m1} has \texttt{id} = 1, and \texttt{mn}, \texttt{n} > 1, id = \texttt{n}.
3. One sender - one receiver (basic scheme).
4. Intruder is modeled following Dolev Yao general intruder model.
5. A Boolean \texttt{flag-s} is set to true if the sender has received the \texttt{im} message.
6. A Boolean \texttt{flag-r} is set to true if the receiver has sent the \texttt{im} message.
7. A Boolean \texttt{received?} is used to check the receipt of a message by the receiver. Since the message is not deleted from the network, when received, the Boolean is set to true, in order not to be received again by the same receiver.
8. The observation \texttt{next} returns the id of the packet to be received by the client.
9. The clock observers are \texttt{now(t)} that returns the time at state \texttt{t}, \texttt{1-sdmin1(t)} that returns the lower bound of sending message \texttt{m1} (\texttt{mn}), \texttt{u-rcvm1(t)} which returns the upper bound of receiving message \texttt{mn}, and \texttt{u-rcvmn(t)} returns the upper bound of receiving message \texttt{m1} at a state \texttt{t}.
10. The constants are: \texttt{init} denotes the initial state, \texttt{d1} is the lower bound of sending an \texttt{m1} message, \texttt{d2} is the upper bound of receiving \texttt{m1} message and \texttt{d3} is the upper bound of receiving \texttt{mn} message. The relation between time delays are declared in the OTS module. The time advancing transition is denoted by \texttt{tick(t,r)}.
11. The non clock values observable from the outside of the protocol are \texttt{nw(t)} that returns the set of messages in the network at a state \texttt{t}, \texttt{ur(t)} which returns the set of random numbers used until state \texttt{t}, \texttt{flag-r(t)} that returns whether the receiver has sent \texttt{im} message, \texttt{flag-s(t)} that returns whether the sender has received \texttt{im} message, \texttt{received?(t,m)} that returns whether message \texttt{m} has been received or not at state \texttt{t}, while \texttt{next(t)} returns the id of the next packet to be received.

The protocol requires an important synchronization assumption, the security condition: The receiver will not accept message \texttt{n} if it arrives after the sender might have sent message \texttt{n+1}, otherwise an intruder can capture message
\(n + l\), and use the key \(k_a\) to fake a message \(n\). This is the reason for using timing constraints to some transitions and the Timed OTS model.

When the sender sends message \(rm\), then receiver can receive it, while also sender can send message \(ml\), since he does not know whether the receiver has already received it (the Boolean received? is not shared between sender and receiver). If the sender sends the \(ml\) before receiver gets the \(rm\), then there exist in the network \(rm\) and \(ml\) with received? values set to false. But in that case there is no problem, since \(ml\) does not reveal a key, while also \(rm\) contains a digital signature. So in that case there is no need for timing constraints other than \(0\) and \(oo\).

But, if the sender sends message \(m_2 (m_{n+1})\) before receiver receives \(m_1\) (and in general \(m_n\)), then the intruder can capture the key that is revealed in the \(m_2\) \((k_1)\), and fake the data part of \(m_1\). This can be avoided if some time constraints are used. So, after sender sends the \(m_1\) message, \(m_2 \(m_n\) should be sent between \(l-sdm2\) \((l-sdmn)\) and \(oo\), \(m_1\) should be received between 0 and \(u-rcvm1\), with \(u-rcvm1 < l-sdm2\). Similarly, the next \(m_n\) message should be sent after the previous has been received. We assume that the delays are constant.

The behavior of the trustable principals is modeled with the corresponding sending and receiving transitions. Each action has an effective condition which is divided into the timing and the non-timing part. There are four transitions modeling receiver and five transitions modeling the behavior of sender. The transitions that have timing part are \(sdm1\), \(sdm2\) and \(sdm3\).

For example, \(sdmn(T, M, I)\) corresponds to that if a message \(m\) of the type \(mn\) with \(id = I, J > 2\), that has been sent by server to client exists in the network, agent server makes the data \(d(a, J+1), f(k(server, client, J+2))\), the key \(k(server, client, J)\) and the message authentication code \(mac(k(server, client, J+1), d(server, J+1), f(k(server, client, J+2)), k(server, client, J))\) and sends it in the message \(mn\) with the id \(J+1\) of the message, providing that \(l-sdmn(T) < now(T)\). The above are specified with equations in CafeOBJ as follows:

```
-- for action sdmn
op c-sdmn : Tesla Msg Int -> Bool
eq c-sdmn(T, M, I) = (M \in nw(T) and mn?(M) and 
  j(M) = I and crt(M) = server and src(M) = server 
  and dst(M) = client and d(M) = d(server, I) 
  and p(M) = f(k(server, client, s I)) 
  and k(M) = k(server, client, I - 1) and 
  mec2(M) = mac2(k(server, client, I), d(server, I), 
  f(k(server, client, s I)), k(server, client, I - 1) and I) 
  and 1 < I < 1 and 
  l-sdmn(T) < now(T) ) .
ceq nw(sdmn(T, M, I)) = (mn(server, server, 
  client, d(server, I), f(k(server, client, I + 
  2))), k(server, client, I), 
  mac2(k(server, client, s I), d(server, I), 
  f(k(server, client, I + 
  2))), k(server, client, I), s I), nw(T)) 
if c-sdmn(T, M, I) .
```

As is shown in the above CafeOBJ code, the observations \(nw\), \(l-sdmn\) and \(u-rcvmn\) change their value after the application of \(sdmn\) in a state \(T\), provided that the effective condition, which is defined by the first equation, holds. The rest observation values does not change.

The delays \(d_1\) and \(d_3\) are declared in the OTS module with relation \(d_3 < d_1\) and define the time order of sending and receiving a message \(mn\).

The intruder tries to glean information from the messages flowing in the network, create and send fake messages based on it. The gleaned quantities are nonces, data, commitments, keys, macs1, macs2, and signatures. The intruder’s fake messages follow the format of the messages of the protocol, in order to be accepted by the receiver. There are 18 transitions modeling the behavior of the intruder. For a fake message of the type \(mn\), the application of transition \(fkmn7\) \((t, k, k', k'', i)\) corresponds to that the enemy fakes \(mn\) \((enemy, server, client, d(enemy, i), f(k, k'), mac2(k', d(enemy, i), f(k, k'), i)\) and put it into the network.

The CafeOBJ equations are:

```
-- for action fkmn7
op c-fkmn7 : Tesla Key Key Key Int -> Bool
eq c-fkmn7(T,K,K',K'',I) = (K \in keys(nw(T)) and 
  K' \in keys(nw(T)) and K'' \in keys(nw(T)) and I > 1) .
ceq nw(fkmn7(T,K,K',K'',I)) = 
  mn(enemy,server,client,d(enemy,i),f(k),K',mac2(K' 
  ,d(enemy,i), f(k),K'),i),nw(T) if c- 
  fkmn7(T,K,K',K'',I) .
```

```
eq ur(fkmn7(T,K,K',K'',I)) = ur(T) .
```
2.4.3 Verifying TESLA Specification

The protocol satisfies the following invariant property (definition taken from the original paper): *The receiver does not accept as authentic any message mi unless mi was actually sent by the sender.*

We have expressed the above property based on our specification as three different invariants:

**Invariant 1.** Whenever the client receives the three messages \( m_r, m_l, m_2 \), i.e.,
\[
\begin{align*}
&f(k_1), n_R, (f(k_1), n_R) \text{ } \text{PK}(S) \text{ and} \\
&d_1, f(k_2), MAC(K_1, d_1, f(K2)) \text{ and} \\
&d_2, f(k_3), k_1, MAC(\ldots),
\end{align*}
\]
then \( m_l \) originates from the claimed source \( S \).

**Invariant 2.** Whenever the client receives the three messages \( m_1, m_2, m_3 \), i.e.,
\[
\begin{align*}
&d_1, f(k_2), MAC(K_1, d_1, f(K2)) \text{ and} \\
&d_2, f(k_3), k_1, MAC(\ldots), \text{ and} \\
&d_3, f(k_4), k_2, MAC(\ldots),
\end{align*}
\]
then \( m_2 \) originates from the claimed source \( S \).

**Invariant 3.** Whenever the client receives the three messages \( m_{n-1}, m_n, m_{n+1} \), \( n > 2 \) i.e., \( d_{n-1}, f(k_n), k_{n-2}, MAC(\ldots) \), and
\[
\begin{align*}
&d_n, f(k_{n+1}), k_{n-1}, MAC(\ldots), \text{ and} \\
&d_{n+1}, f(k_{n+2}), k_n, MAC(\ldots),
\end{align*}
\]
then \( m_n \) originates from the claimed source \( S \).

The above invariants are expressed in CafeOBJ notation in a module called inv.mod as operators inv1, inv2 and inv3 respectively. To prove the most important and general property inv3, we used five more invariants as lemmas that we had then to prove. In general, to prove the three invariants that constitute the basic property of TESLA protocol, we used 29 invariants. Most of them were state invariants, while there were also lemmas on data types, such as Network.

Two invariants that were used as lemmas and include timing information are inv8 and inv12:

For any reachable state \( T \) and any message index \( N \),
\[
\begin{align*}
\text{eq inv8}(T, N) &= N > 1 \text{ and } 1 - \text{sdmm}(T) \leq \text{now}(T) \text{ and} \\
\text{mn}(server, server, client, d(server, N), f(k(server, client, (N + 1))), k(server, client, (N + 1))), k(server, client, (N + 1))), k(server, client, (N + 1))), N)
\end{align*}
\]

\[
\begin{align*}
\text{Ceq fkmn7}(T, K, K', K'', I) &= T \text{ if not c-} \\
\text{Ceq u-rcvmn}(fkmn7(T, K, K', K'', I)) &= u-rcvmn(T).
\end{align*}
\]

The former says that if an original message \( mn \) exists in the network in a state \( T \) with \( n > 1 \) and \( 1 - \text{sdmm}(T) \leq \text{now}(T) \), then the message has been already received by the client. The latter says that if an original message \( mn \) exists in the network in a state \( T \) with \( n > 1 \) and it has not yet been received by the client, then \( u-rcvmn(T) < 1 - \text{sdmm}(T) \).

Apart from the lemmas, the proof scores written include exhaustive case analysis. In general, the verification of TESLA specifications follow the same principles and methodology as the Standard OTS and some representative code fragments can be found in [26].

3. Lessons Learned

3.1 Introduction

Writing algebraic specifications and verifying them with the CafeOBJ system in the framework of the OTS/CafeOBJ method, has the advantage of a simple underlying theory, since it is based on equational logic. We believe that equational logic is easier to learn than other logics such as higher order logic, because replacing equals by equals is used as everyday life reasoning. Hence, we state that interactive theorem proving with the OTS/CafeOBJ method is easier to learn than those with other methods.

The two basic tasks that a specification engineer has to perform to specify and verify a system is the system and property description. Both suppose a deep understanding of the system/protocol. In many cases, an incorrect system’s specification may lead to unsuccessful verifications, which implies specification revisions/updates and verification retries. As a result, it is difficult for inexperienced users to do interactive theorem proving, especially that complex systems enjoy desired properties with the OTS/CafeOBJ method.

We believe that our experiences gained and lessons learned from the TESLA verification experiment that are presented below, can help such users learn effectively interactive theorem proving with the OTS/CafeOBJ method.

In the case of TESLA protocol, which is a complex protocol with timing constraints, many verification retries and specification revisions were performed. The reason was basically protocol functions misunderstanding, that lead to wrong descriptions. For instance, at the initial steps of our work on TESLA, we made three fundamental errors:

- A first attempt to model TESLA [27] was unsuccessful,
since we did not expressed correctly the timed part of the specification as a Timed OTS. As a result the expression of the invariant properties was not representative of the protocol’s critical properties.

- Next, we tried to model the protocol without timing constraints [28], only by adding some index to each packet. But when we tried to express the security condition, we found out that a counterexample was obvious, without adding the timing constraints: An intruder could steal a message and put into the network with altered data. Then the receiver could not identify the faked from the original message.

- The third error was related to the expression of the basic property, during the verification of our Timed OTS model of the protocol [26]. During the process of verifying/writing proof scores, we realized that without a boolean observation received? in our property expression, a message could exist in the network without having been received by the receiver, which was not representative for the protocol behavior.

Some less important but necessary modifications of the specification of our Timed OTS model that we had to do was:

- Initially we did not use an observation next to model the id of the next packet to be received by the client. As a result the client could accept a faked message that existed in the network with smaller id than the last received.

- Without a special transition sdm2 for the sending of the second message m2, it was not possible to model the effective condition c-sdmn, n=2, since sdmn, n=2 depends on the existence of message m1 in the network which has different format from mn, n > 1.

Next, we give a detailed description of each error.

3.2 Attempts to Formally Analyze TESLA

3.2.1 Initial Attempt

Our first attempt to model TESLA was [27]. The TESLA model contained some timing information but did not follow the Timed OTS model. The property of the specification that we proved was:

At any reachable state, if a key can be obtained by the enemy, then either the key belongs to the enemy or it has been revealed as part of a message

The above property is important for the protocol, but is not equivalent to the original correctness property, which says: The receiver does not accept as authentic any message unless it was actually sent by the sender.

The first property probably can be used as a lemma to discard some subcase, but does not express the original correctness property of the protocol.

Additionally, the way it was expressed does not follow the Timed OTS model

\[
\text{inv1}(T, K) = \not\exists \text{in keys}(\text{nw}(T)) \implies (p(K) = \text{enemy}) \text{ or } (s \in i(K)) \ast d \leq \text{now}(T).
\]

In general, the specification of the system was simpler than the final presented in this paper, and did not describe all aspects of TESLA protocol.

3.2.2 Second Attempt - No Timing Constraints

In our second attempt we thought of making things simpler by getting rid of time, and instead of it, to use indexing at each packet.

More specifically, among other differences between the model presented in this paper and that of the second attempt, has to do with expression of action sdmn:

So, sdmn(t, p, m, i) corresponds to that if a message m with id = i - 1, i > 1, that has been sent by p exists in the network, p makes the pseudorandom function f(k(p, i+1)), the key k(p, i-1) and the message authentication code mac(k(p, i), f(k(p, i+1), k(p, i-1)) and sends it in the message mm with the id i of the message. The main difference between this expression and the expression presented in page 4 of this paper is the timing condition l-sdmn(t) <= now(t).

During the verification of the most important property of our model, that was expressed as:

At any reachable state, a packet can be authenticated if and only if there exist in the network the preceding and succeeding packets and their real creator is not the intruder.

A counterexample was obvious: A malicious agent could alter a message index based on the information that exists in the network, and put into it fake messages that the receiver cannot distinguish as fake.

3.2.3 Third Attempt

Our third attempt, which resulted to the model presented in this paper, had some intermediate steps that provided to us a better understanding of the protocol and of the Timed OTS model, as a result of verification failures and corresponding specification/model revisions:

- Error 1: Invariant property expression

The expression of the invariant property plays a key role to the verification of a protocol/system. In our case, many revisions of the basic protocol property expression were done. For example, initially we declared the protocol’s agents as variables in the expression of the invariant property. That gave an increased complexity to the proof, since we had to analyze more subcases, without an obvious reason. While basic TESLA protocol assumes one sender - one receiver, we simplified the expression by declaring protocol’s agents as server and client constants.

Another, more essential modification that we did to the expression of the property was the addition of a boolean observation received? that takes a state and a message and returns whether the message has been received at that state.

The invariant before:

\[
\text{inv}_{<}(x, \text{server}, \text{client}, \ldots, \text{data}_{a-1}, \ldots) \in \text{nw}(t) \land \\
\text{inv}_{>}(y, \text{server}, \text{client}, \ldots, \text{data}_{a}, \ldots) \in \text{nw}(t) \land
\]

\[
\text{inv}_{=} (x, \text{server}, \text{client}, \ldots, \text{data}_{a-1}, \ldots) \in \text{nw}(t) \land \\
\text{inv}_{<}(y, \text{server}, \text{client}, \ldots, \text{data}_{a}, \ldots) \in \text{nw}(t) \land
\]

\[
\text{inv}_{=} (x, \text{server}, \text{client}, \ldots, \text{data}_{a-1}, \ldots) \in \text{nw}(t) \land \\
\text{inv}_{>}(y, \text{server}, \text{client}, \ldots, \text{data}_{a}, \ldots) \in \text{nw}(t) \land
\]
\[ m_{n+1}(z, server, client, \ldots, data_{n+1}) \text{ in } nw(t) \Rightarrow m_{n}(server, server, client, \ldots, data_{n}) \text{ in } nw(t) \]

while the invariant after:

\[ m_{n-1}(x, server, client, \ldots, data_{n-1}) \text{ in } nw(t) \wedge \]
\[ received?(t, m_{n-1}(x, server, client, \ldots, data_{n-1})) \wedge \]
\[ m_{n}(y, server, client, \ldots, data_{n}) \text{ in } nw(t) \wedge \]
\[ received?(t, m_{n+1}(z, server, client, \ldots, data_{n+1})) \Rightarrow \]
\[ m_{n}(server, server, client, \ldots, data_{n}) \text{ in } nw(t) \wedge \]
\[ received?(t, m_{n}(server, server, client, \ldots, data_{n})) \]

Without the boolean \( received? \), a message could exist in the network without having been received by the receiver. In that case, the property could not be proved.

- Error 2: Action operator \( sdm2 \)

Initially we did not use a special transition \( sdm2 \) to model sending of a message of type \( mn \), \( n = 2 \) since we captured the behaviour by \( sdmn \) transition.

But when tried to express the effective condition of \( c-sdmn \), we realized that it should be different for \( n = 2 \) and for \( n > 2 \) due to the existence of message \( m1 \) in the network, that follows different format. So, we defined a special action operator \( sdm2 \) for \( n = 2 \), and \( sdmn \) for \( n > 2 \).

- Error 3: Observation operator \( next \)

During the process of verification and corresponding specification revisions, we realized that if an observation that represents the id of the next packet to be received by the receiver did not exist, the client could accept a faked message that existed in the network with smaller id than the last original message received.

To overcome the above, we added the observation \( next \) that takes a state as argument and returns the id of the packet to be received by the client. A diagrammatic animation of the specification that was used is as follows:

We assume that at state \( t0 \) there exist original messages \( m2, m1, rm \) and \( im \) into the network and the client has received messages \( m2, m1, rm \), while server has received message \( im \).

At state \( t1 \), the sender puts into the network message \( m3 \). The other participants do nothing.

At state \( t2 \) the enemy gleaned data from the network, forms the fake message of the type \( m2 \), called \( fkm2 \), and puts it into the network.

Without using the observation \( next \), the client in the next state \( t3 \) may receive every message that exists in the network, which is not correct for our protocol, since the client may be a user of Internet TV and this may result in long delays.

With observation operator \( next \), when the message \( m2 \) was received, then \( next \) would be set to 3, and the effective condition of \( t3 \) would contain \( next(t3) = 3 \). So the client would receive the right message \( m3 \).

As someone can easily infer, the above errors in the specification of the protocol may vary with specifier, and/or the system to be described. But the lesson learned from this experience, is general and refers to the need for revisions and corrections to the specification during the process of the formal analysis of a system. This result may be applied not only to the OTS/CafeOBJ method but to any other formal method.
3.3 Summary of Experiences and Lessons Learned

To sum up our results, we place in order the experiences gained and the lessons learned:

3.3.1 Experiences

Several revisions need to repeat until both systems and properties are specified as intended, especially if systems are complex such as TESLA and properties are non-trivial such as those verified in the case study.

3.3.2 Lessons

Lesson 1. When you reach a level such that you think that it is possible to specify systems and properties, you may want to do so even though your understanding is not profound. This is because it is almost impossible to understand complex systems and non-trivial properties in an initial attempt. But as the understanding of the system is not profound, errors will exist in the specification, and it is indispensable to repeat revisions of system and property specification. In an interactive process, repetition of revisions of system and property specifications helps you get better understanding of systems and properties, and vice versa.

Lesson 2. You may want to use diagrammatic systems specification animations by hand, symbolic systems specification animation by CafeOBJ and verification attempts by writing proof scores in CafeOBJ, to get better understanding of systems and properties.

Getting better understanding of systems and properties, and revisions of system and property specifications, repeat until you understand systems and properties profoundly and then system and property specifications are as intended.

Although the verification technique used in the case study is interactive theorem proving, we think that the above lessons learned are also useful as well as meaningful for other verification techniques such as model checking.

3.4 Some Statistics of the Case Study

The specification of the protocol, together with the file which describes the invariant properties in CafeOBJ is about 1400 lines of specification code, which is a reasonable number.

As we have already mentioned, to carry out the proof we have proved 29 different invariant properties. Each of the 29 invariants needs a different number of proof fragments called proof scores. Totally, the proof is about 35000 lines of code long, but with many reusable parts.

In general it is difficult to estimate the workload for specifying and verifying TESLA. Firstly, it depends on the way of working the case study. In our case, we worked in a less than part time way, with some intermediate pauses. The total amount of time to finish the case study was a couple of years, but the actual amount of time was much less and cannot be estimated.

The above time space can be distributed into the following activities in a percentage estimation:
- 10% of time for reading and understanding the Timed OTS/CafeOBJ method, assuming knowledge of the OTS/CafeOBJ.
- 20% of time for studying the protocol.
- 70% of time for specifying and verifying the protocol. We don’t separate specification/verification because we consider them as one. This is because we revised many times the specification as a consequence of verification and better understanding of protocol’s functionality.

4. Related Work

TESLA protocol has been formally specified and verified in three different works [15]–[17]. In [15], the protocol is analysed using TAME [18], an interface to PVS [19] specialized for proving properties of automata. In this approach the system is first modelled as an LV timed automaton [20], next any desired system property is expressed as a state invariant and finally, the validity of the state invariant is established by developing auxiliary invariants that supports its proof. Both our approach and this approach belong to the theorem proving family, but the main difference between them is that in the case of timed automata you should identify all the states involved in the real time system in advance, which may be difficult. On the contrary, you have not to explicitly identify states involved in a real time system in advance to model the system as a TOTS.

In [16], a CSP [21] finite model of TESLA is model checked using the FDR [22] model checker. The authors’ challenge was to apply model checking to such an infinite system. A number of reduction strategies were developed and incorporated into the model to keep state space within a feasible range. They have also extended their model to capture the Scheme II of the protocol that involve modelling of unbounded hash-chains. Synchronization between sender and receiver processes in this approach is captured by introducing a special event *tick* that represents the passage of one time unit. This synchronization allows the receiver to tell whether it has received message $n$ before sender might have sent message $n+1$. 
Finally, in [17], the authors present the application of an extension of a model checker for multi agent systems called MCMAS-X to the verification of TESLA. The model of the protocol is written in an SMV-like programming language called ISPL which is based on TDL [23] temporal epistemic logic.

All approaches, including ours [26], take the simple case of one sender – one receiver, but is straightforward to extend the model to capture the most complex cases.

The main difference between the above works and this paper, is that we present detailed experiences and lessons learned by the case study conducted, which is the main contribution of the paper.

Generally, formal verification of real time systems has been studied by many researchers. Another OBJ language that has a real time extension is Maude [24], with Real-time Maude [25]. The main difference between our approach and that of Maude, is that the system to be analyzed with Maude should have finite state space.

Reported lessons learned by conducting case studies using formal methods are not met very often in bibliography. In [29]–[32] the authors apply different formal methods to the specification and verification of industrial systems and they present their experiences.

5. Conclusion and Future Issues

We have presented the TESLA verification experiment and the lessons learned by this. It is also the first time that the Timed OTS/CafeOBJ method is applied to such a complex protocol/system.

The lessons learned reported in this paper shows that the method we used provides a better understanding of the protocol to be analyzed than the automated methods. This is very important not only for the researcher that tries to understand an existing protocol, but also for the designer of a protocol.

The experiences reported in this paper may be similar to old wisdom of formal methods that have been found by using some specific formal methods such as Z and B. But here, we have shown that these experiences are not only specific to some class of formal methods such as Z and B method, but also to an algebraic specification and verification technique, which can be though as a new wisdom.

In addition, the specific errors we have encountered in the case study must depend on human beings. But, we think that as one applies a formal method to a non-trivial problem, he/she must encounter some non-trivial errors that are similar to those we have encountered. This is why we think that our experiences and lessons should be useful to tackle the errors he/she encounters.

Even though our lessons are not very generic and cannot be applied to any case, we think that it is very important to accumulate as many lessons as possible, which is needed to facilitate use of formal methods. In this sense, we think that our experience and lessons described in the paper are valuable.

In general, it is not straightforward to get a reasonably good understanding of a complex system such as TESLA in a first attempt, as shown in our experience gained. We believe that since system specification requires several revisions according to better understanding of target systems, it would be very useful to have some means and tools to help human users get better understanding of target protocols/systems, such as diagrammatic animations or/simulation tools.

Diagrammatic systems specification animation can be effectively used to get better understanding of systems and properties. Interactive theorem proving also requires to get better understanding of systems so that good lemmas can be conjectured. But, our current practice uses it by hand. Some systematic way to animate diagrammatically system specifications (including a tool supporting it) is one possible future issue.

Additionally, since model checking automatically finds counterexamples showing that systems specifications do not enjoy properties if the counterexamples are located at reasonably shallow positions, model checking may facilitate getting better understanding of systems and properties. Real-time Maude [25] is an extension of Maude that is a sibling language of CafeOBJ and has model checking functionalities as Maude. Although there is an attempt to combine Maude model checking functionalities and the OTS/CafeOBJ method [14], no attempt to combine Real-time Maude model checking functionalities and the Timed OTS/CafeOBJ method has been made. Hence, this is another possible future issue.

Finally, another idea on saving time when dealing with complex real life systems and protocols, is to develop libraries of reusable similar modules used to real time, security protocols, etc.

Acknowledgments

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS.

References


