2010 Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement Companion

Utilization of Timed Automata as a Verification Tool for Security Protocols

Ahmet Koltuksuz
Yasar University
Department of Computer Engineering
Izmir, Turkey
ahmet.koltuksuz@yasar.edu.tr

Burcu Kulahcioglu, Murat Ozkan
Izmir Institute of Technology
Department of Computer Engineering
Izmir, Turkey
{burcukulahcioglu, muratozkan}@iyte.edu.tr

Abstract— **Timed Automata is an extension to the automata-theoretic approach for the modeling of real time systems that introduces time into the classical automata. It has become an important research area in both the context of formal languages and modeling and verification of real time systems since it was proposed by Alur and Dill in the early nineties. Timed automata proposes an efficient model checking method for verification real time systems having mature and efficient automatic verification tools. One of the application areas of timed automata is the verification of security protocols which are known to be time sensitive. This study aims to make use of timed automata as a verification tool for security protocols and gives a case study on the initial part of the Neuman-Stubblebine Repeated Authentication Protocol.**

Keywords-timed automata, model checking, security protocol verification

I. INTRODUCTION

Real time systems can be analyzed using formal methods to verify that a system meets some specified requirements. In the literature, most verification methods involve the partial ordering of the occurrence of events in a qualitative notion instead of modeling quantitative time information. However, the correctness of a real time system depends on its quantitative timed properties.

To meet the need for timed formalisms, some untimed formalisms are extended with timing information such as timed Petri nets [1], many real time logics [2] and timed process algebras such as CSP [3]. However, timed automata [4] is the most commonly used model for timed systems having mature and efficient automatic verification tools and for an easily understandable syntax and semantics with the support of C-like data structures.

Timed automata is proposed as an extension to the automata-theoretic approach, which is extended with clock variables. Timed automata theory has become an important research area and been widely studied in the context of both formal languages and verification of real time systems.

The theory of timed automata allows us to create models of real time systems which can be verified using model checking methods [5]. Model checking with timed automata involves building a finite model of a system and verifies a property by traversing through all reachable states. It has the advantages of being fully automatic, and generating counter example in case of a negative result nevertheless, it suffers from the state space explosion problem.

One of the most important application areas of timed automata is the verification of the security protocols. Since the use of computers and the internet is considerably increasing, the correctness of security protocols is getting more important. Since an attacker can exploit the timing of message flows, quantitative time information is critical for security issues. This study utilizes timed automata as a verification tool for security protocols including timing information. We directly model the Neuman-Stubblebine repeated authentication protocol [6] using the UPPAAL timed automata tool [7] and perform verification by analyzing its security properties to find possible attacks on it.

The next section gives the related work including the timed automata studies on security protocols. Section 3 briefly defines timed automata, the data structures used in its implementation, and the UPPAAL tool. Section 4 explains the modeling of the initial authentication part of the protocol including the modeling of cryptology, automata for the protocol principals and the intruder. The verification of our model is performed in Section 5, in which we present the type flaw attack we found and analyze the quantitative timing properties of the protocol. In addition, we give comments on the modeling and verification of the subsequent authentication part. Section 6 concludes the paper with the results we obtained and the further perspectives for the analysis of the subsequent part of the protocol.

II. RELATED WORK

Timed automata has several academic and industrial case studies such as the modeling and verification of TDMA (Time Division Multiple Access) protocol [8], audio-video protocols [9], a power controller [10] and a lip synchronization algorithm [11]. In this study, we focus on modeling and verification of security protocols.

In the literature, several theorem proving and model checking methods are used to verify the correctness of security protocols, most of which involve the qualitative notion of time rather than the quantitative notion. In this paper, we concentrate on the timed automata formalism verified with model checking methods.

Some recent studies analyze security protocols with quantitative timing properties involving the use of timed automata. The studies in [12] and [13] examine Kerberos, TMN, Neumann Stubblebine, Andrew Secure and Wide Mouthed Frog protocols by not modeling them directly as timed automata, but translating a language specification of a

978-0-7695-4087-0/10 $26.00 © 2010 IEEE
DOI 10.1109/SSIRI-C.2010.27

security protocol automatically to timed automata without integer variables. Then, translated timed automata is used as input for the model checker KRONOS [14] and VerICS [15]. Similarly in [16], a model checking tool is presented which translates a security specification language into timed automata and uses the UPPAAL tool as the verification engine. Additionally, a case study on Wide Mouthed Frog protocol is provided.

Similar to our case study, these studies perform verification using timed automata tools. However, our approach is closer to the studies in [17] and [18] which model Needham-Schroeder and Yahalom protocols directly with timed automata. Directly modeling provides us full control over the timed automata model and enables us to make use of the full expressiveness and data structures of UPPAAL. Moreover, we are not required to have an expertise on a specification language to model a protocol.

III. TIMED AUTOMATA

Timed automata is proposed by Alur and Dill [4] in the early nineties as an extension to the classical automata. It is equipped with a number of real-valued clock variables which record the passage of time since they have been reset. All clocks are synchronized and they run at the same speed. In a timed automaton, it is assumed that a transition from one state to another is assumed to be instantaneous; in other words, time passes only in states, not on edges.

A. Timed Automata Theory:

Timed automata has an extensive theory in the context of formal languages. Besides the first proposed model, some variants of the model are also proposed and analyzed for some decidability problems. In this paper, we give a brief introduction to the classical timed automata model which is used in the implementation of the timed automata tools.

Definition 1. A *timed automaton* is a tuple $< \Sigma, S, S_0, S_F, C, E >$ where

- $\sum$ is a finite event alphabet
- S is a finite set of states
- $S_0 \subseteq S$ is a set of start states
- $S_F \subseteq S$ is a set of final (accepting) states
- C is a finite set of clocks
- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C),\ \phi \in \Phi(C)$ are the edges where $\phi := x \leq k \mid k \leq x \mid x < k \mid k < x \mid \phi \wedge \phi \mid \phi \vee \phi$ with $x \in X,\ k \in \mathbb{N}$.

In this definition, ϕ is the set of clock constraints. A clock constraint can be a guard on an edge to control if it is allowed to take the transition in the current time or can be associated with a location and called *location invariant*.

Fig. 1 gives a timed automaton drawn using UPPAAL, where the state with double border line is the initial state. This automaton has two clock variables x and y; and the clock constraints "$x > 3$", "$x < 10$" ,"$y = 9$" as the guards on edges. At the transitions, the clock valuations can be tested and a set of clocks can be reset. To take the edge from s_0 to s_1, event a must be received and x must have a value greater than 3. If this transition is enabled, x is reset. The constraint $x < 7$ is an invariant and forces to take the edge from s_0 to s_1 when x has a value smaller than 7. Note that this invariant does not have the same effect as having clock constraint "$x > 3 \wedge x < 7$" on the transition.

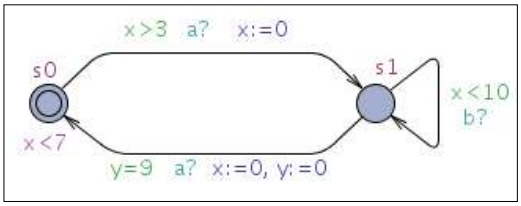

Figure 1. An example timed automaton

A run of timed automaton A has the form: $(s_0, v_0) \xrightarrow{t_0} (s, v'_0) \xrightarrow{a} (s_1, v_1) \xrightarrow{t_1 - t_0} (s_1, v'_1) \xrightarrow{a_1} \dots\dots \xrightarrow{a_{n-1}} (s_n, v_n)$ where each pair (a, t) is a timed event with $t \in \mathbb{R}$, which is the timestamp of the event $a \in \Sigma$. A run is an accepting run if $s_n \in S_F$. For example, the timed word for Fig. 1 is $s_0 \xrightarrow{\delta(3.8)} s_0 \xrightarrow{\delta(a)} s_1 \xrightarrow{\delta(2)} s_1 \xrightarrow{\delta(b)} s_1 \xrightarrow{\delta(3.2)} s_1 \xrightarrow{\delta(a)} s_0$.

A certain property is *decidable* for a formal language if there is a procedure that can determine whether the property holds or not in the model. For timed automata, *emptiness*, *universality* and *language inclusion* problems are the most studied decision problems since they are also the fundamental problems for verification. Timed automata has decidable emptiness and undecidable universality and language inclusion problems.

Definition 2. *Emptiness problem* is the problem of "given a timed automaton A, is the set of timed traces of A empty?".

Emptiness problem is fundamental for verification tasks since it is reducible to the *reachability problem* that tests whether a state can be reached in a model. In a verification task, given an implementation and a specification, reachability problem is used to test whether a state which satisfies the specification is reachable in the implementation. However, the configurations of timed automata are infinite and naive explicit state search is not possible. The decidability of the emptiness problem for timed automata is proved [4][19] by constructing a model on which finite state analysis can be performed. A *region automaton* that mimics the runs of the timed automaton is constructed so that the emptiness problem of a timed automaton can be examined by checking the emptiness of its region automaton. However, it is shown that the problem is PSPACE-complete since the number of regions is exponential in the number of clocks of the timed automaton.

Theorem 1. The problem of deciding the emptiness of the language of a timed automaton A, is $PSPACE - complete$.

B. Implementation of Timed Automata

Similar to the other model checking methods, it is needed to perform reachability analysis on the model to perform a verification task using timed automata. Thanks to the decidability of the reachability (and in turn, emptiness) problem, we can perform model checking by traversing the state space for reachability testing. However, the region automaton is not feasible to implement since it suffers from a combinatorics explosion. For this reason, symbolic representation of states and on-the-fly model checking are preferred resulting in considerable space and time savings.

In the implementation of timed automata, *zones* which can be efficiently represented using *Difference Bounded Matrices (DBM)* are used instead of *regions* and symbolic reachability analysis is performed on these data structures.

Timed automata tools use on-the-fly reachability algorithm that calculates the states on-the-fly rather than pre-computing. Thus, only the needed part of state space is computed. The use of the symbolic structures and symbolic model checking algorithms make timed automata to be implemented in an efficient way. [20] [21]

C. A Timed Automata Tool - UPPAAL

UPPAAL [7] is a freely available timed automata tool that provides an integrated tool environment for modeling, validation and verification of real-time systems. It has a graphical user interface for modeling, a simulation tab, and a verification engine for automatic verification of specifications. It is an efficient and mature tool used in several case studies [8][9][10][11] which is in continuous development. In this study, UPPAAL 4.0.10 is used.

UPPAAL extends timed automata with C-like data types such as integers, arrays and functions. It allows using urgent and committed states that ease modeling of a system.

UPPAAL verification engine uses a subset of CTL (Computation Tree Logic) as the specification language, consisting of *state formulae* or *path formulae* that can be classified into reachability, safety, and liveness properties.

IV. Modelling Neuman-Stubblebine Authentication Protocol

A. Protocol Modeling with Timed Automata

Timed automata model a system as a *network of timed automata* which is composed of several components each having a transition system. It consists of some number of timed automata running in parallel that may communicate and synchronize on some events. In a network of timed automata, the events are partitioned into the set of output and input actions. The output statement over channel a is labeled as $a!$ (*emission*) and an input statement over channel a is labeled as $a?$ (*reception*). Two edges in different processes can synchronize if one is emitting and the other is receiving on the same channel. In the execution of a network of timed automata, the transitions of the timed automata with a shared action are synchronized and the transitions that does not correspond a shared action are interleaved.

The timed automata model for a protocol is generated by building a finite state machine whose states and transitions simulate the behavior of a protocol run. To find an attack on the protocol, all possible states are explored and analyzed if the protocol has some security flaws.

The protocol is modeled as a network of timed automata composed of the initiator, responder, server and the intruder automata. These principals communicate with each other by using synchronization channels and shared variables. For example when the initiator emits the $init\_msg!$ signal over $init\_msg$ channel, this means that it has created the message and the message is assigned to the global message variable msg which is shared between the principals. The network takes this message over $init\_msg?$ and emits $resp\_msg!$, which is captured by the responder. Then, the responder reads the global message variable msg.

B. Modeling Cryptology

In our model the cryptosystem is assumed to be perfect, so we used an abstraction for the cryptographic operations. These cryptographic abstractions are held in the local functions $gen\_nonce()$, $encrypt(int\ plaintext, int\ key)$ and $decrypt(int\ ciphertext, int\ key)$ of each principal. When these functions are called, the result of the operation is assigned in their local variable $result$. For these time consuming operations, we make use of the timed automata that can model the delay and deadline requirements.

A message sent/received by an entity is represented as an integer, which contain the information described in the protocol specification. The creation of the messages and the encryption/decryption scheme we used are similar to the model used in [17] and [18]. Before moving on the creation of a message, let us examine how to model the nonce generation, encryption and decryption.

1) Nonce Generation:

The nonce is generated by calling the $gen\_nonce()$ function which returns a result by incrementing the global $nonce$ variable.

2) Encryption/Decryption:

Two arrays $plain$ and key are used for encryption and decryption, where the first one holds the plaintexts and the latter holds the keys. When a block is encrypted, the plaintext is placed in the $plain$ array and the key is placed in the key array. Then, the corresponding index is returned as the ciphertext, which is the result of the encrypt operation. A plaintext can be decrypted only if the given key is same with the key in the key array corresponding to the element in index (ciphertext) to be decrypted.

3) Representing Protocol Variables:

A protocol message is represented as an integer. UPPAAL uses 16 bit integers where the leftmost bit is the sign bit. In order to contain the whole message in an integer and have simplicity in the model, we have to limit the number of bits to represent the blocks contained in a

message. In our model, the nonces, keys and indexes (or ciphertexts) are represented by 4 bits, the agent ids and t_b is represented by 2 bits. In the implementation, the possible values for these variables are restricted since there may be some problems in the model when these values coincide.

To reduce the state space of the intruder automata, we used only two length variables $Length1 = 4$, $Length2 = 2$ which are the length of bits to represent a variable.

4) Creating and Reading Messages:

A message is created by using *shift,* and or operations. Parts of the message are merged by shifting the message to the left as the length of the part to be appended. Then, the new part is appended using the or operation.

To extract information from a received message, *shift* and and operations are used. This time the message is shifted to the right and the and operation with the mask is applied.

For example, when the initiator creates the message A, N_a, it shifts A left for $Length1$ times, and or s the result with N_a. To read this message, the responder and s it with the mask to obtain N_a, and shifts left $Length1$ times and applies and operation to obtain the claimed identity.

C. Neuman-Stubblebine Repeated Authentication Protocol

Neuman-Stubblebine protocol [6] is a repeated authentication protocol that provides mutual authentication between two principals. It consists of two parts. First, the initial authentication part is executed which provides mutual authentication. In this part, the initiator acquires a ticket to be used in the subsequent part of the protocol. The subsequent part is used to re-authenticate the principal identities without using the server. This part can be repeated several times until the ticket expires.

In the protocol specification given for the initial and subsequent authentication parts, A , B and S are the principals where A is the initiator, B is the responder and S is the key distribution server. K_{as} , K_{bs} , and K_{ab} are the shared keys where the subscript letters denote the principals whom the key is for (e.g. K_{as} is shared between A and S). $\{X, Y\}_k$ means, X concatenated with Y, encrypted with k.

1) InitialAuthentication Part:

The initial part requires the exchange of four protocol messages. A initiates the authentication by sending its identity A and a nonce N_a. After B receives this message, it sends its identity and a nonce created by B as clear text and A's name, nonce and a suggested expiration time for the credentials as a block encrypted with the key K_{bs}. The server can decrypt this message since it knows K_{bs}, and assures that they are created by B. Then, the server sends A a ticket, and B's nonce. It also sends the identity of B, A's nonce, a session key K_{ab}, the expiration time t_b encrypted with K_{as} . A decrypts the block encrypted with K_{as} and verifies the N_a is same with the N_a in message 1. In the last message, it sends the ticket and N_b to B, proving its identity.

1. $A \rightarrow B: A, N_a$
2. $B \rightarrow S: B, \{A, N_a, t_b\}_{K_{bs}}, N_b$
3. $S \rightarrow A: \{B, N_a, K_{ab}, t_b\}_{K_{as}}, \{A, K_{ab}, t_b\}_{K_{bs}}, N_b$
4. $A \rightarrow B: \{A, K_{ab}, t_b\}_{K_{bs}}, \{N_b\}_{K_{ab}}$

This initial authentication provides mutual authentication between the principals. After this initial part, the initiator A possesses the ticket $\{A, K_{ab}, t_b\}_{K_{bs}}$ and the session key K_{ab} that can be used for subsequent authentications.

2) Subsequent Authentication Part:

In this second part, A uses the ticket to authenticate itself to the responder. B checks the sender's identity, shared key and the expiration time of the ticket. If it is valid, the authentication is provided between the principals.

5. $A \rightarrow B: N'_a, \{A, K_{ab}, t_b\}_{K_{bs}}$
6. $B \rightarrow A: N'_b, \{N'_a\}_{K_{ab}}$
7. $A \rightarrow B: \{N'_b\}_{K_{ab}}$

In our experimental part, we give the timed automata model and the verification results for the initial authentication part of the protocol. Then, we will comment on the modeling and verification of the subsequent part.

D. Timed Automata Model for Neuman-Stubblebine Initial Authentication

1) Initiator, Responder and Server

The knowledge bases of the principals are modeled using local variables for each automaton. For example, the initiator A has K_{as} as initial knowledge. Then, it generates N_a, gets a ticket, learns K_{ab}, and t_b which will be added to its knowledge base. The principal B, has K_{bs} as the initial knowledge and gets a claimed id, N_a, generates N_b, t_b and learns K_{ab}. It should keep this knowledge to use in the later steps of the protocol e.g. while checking the values received in the fourth step of the message.

In order to be able to analyze timing properties, each principal automaton has its local clock variable to keep the time elapsed for the cryptographic operations, in addition to a global clock representing the total time passed.

The Initiator automaton; which is given in Fig. 2, is activated by the $Init$ automaton that emits the $start!$ signal. The Initiator, first generates a nonce by calling its local function. Time can elapse during the operation and when the nonce is generated it is written in the $result$ variable. Note that we make use of the committed states (labeled with "C") that allow modeling of atomic behaviors and avoids any unnecessary interleaving in the model.

It creates the message A, N_a by assigning the message to the global variable msg. Since this value is global, it can be read by other principal's automaton. Then, it signals $init\_msg!$ to indicate that it has sent the message. This signal is captured by the network and transmitted to the responder. After sending the message, in state $A5$, it waits for the protocol message 3. When it is sent by the server, network signals $init\_msg!$ which will be captured by the $init\_msg?$ of the initiator that brings it to state $A6$. In this transition, initiator extracts the block encrypted with K_{as}, gets the ticket which is $\{A, K_{ab}, t_b\}_{K_{bs}}$, and N_b. It decrypts

the block with the key K_{as} which is shared by the initiator and the server. Similar to generating a nonce, decryption of a message is performed by its function. The guard on the transition from $A6$ to $A7$ guarantees that B's identity sent by the server is same by the identity that A wants to communicate with, and the nonce value is same with the one generated by itself. If this guard is satisfied, it gets the K_{ab} and t_b. It encrypts N_b with K_{ab} and creates the message $\{A, K_{ab}, t_b\}_{K_{bs}}, \{N_b\}_{K_{ab}}$ by concatenating the ticket and the encrypted block. After sending this message, the initial protocol execution finishes for the initiator and it sets its local variable $finish1$ to 1.

Let us assume that A has received a wrong message from S in the third step. Then, N_a will be different from the one A itself generated, the guard will not be satisfied and the transition from state $A6$ to $A7$ will not be taken. Hence, the automata will deadlock and the $finish1$ value will be 0 which means that there is something wrong with the execution of the protocol.

The automata for the Responder (see Fig. 3) and the Server are modeled in a similar way.

2) Dolev-Yao Intruder

The flaws of a security protocol are examined by modeling an *intruder* who wants to exploit the features of a protocol. In our study, we use the Dolev-Yao intruder [22] which has the full control of network and has the abilities to deliver or intercept messages, decompose messages, do encryption/decryption and compose fake messages.

As it is seen from Fig. 3, that the intruder can behave as a simple network which only receives and transmits received messages. In addition, besides the correct recipient, it is possible to send a message to any principal that the intruder wants (transitions between states $I1$ and $I3$).

The Dolev-Yao intruder model allows the use of the knowledge of the intruder which includes the identities of the agents, its own keys and nonces, every messages it received, every part of the messages it received, everything it can generate by encrypting or decrypting something and every concatenation of data it knows.

The Dolev-Yao intruder can capture the packets, decompose into its constituent parts and examine them. For example, when the message A, N_a is captured by the intruder, it has the ability to read the initiator's identity and its nonce N_a and add them to its knowledge base. Hence, after receiving a sent message, we use the piece of timed automata (with states $I5$ and $I6$) to enable the intruder improve its knowledge base adding the information extracted from the messages sent. In fact, the intruder can nondeterministically take one of the transitions from $I5$ to $I6$ to read a message. However, this makes the state space grow enormously. Because of that, we used some guards to limit the possible number of transitions. It is important that these limitations do not lessen the power of the intruder, but decreases the number of infeasible executions.

The intruder has the ability to generate a nonce, do encryption and decryption (between the states $I6$ and $I14$) using the parameters in its knowledge base. It again nondeterministically selects the parameters to apply encryption/decryption. To avoid state space explosion [23], we used guards that allow using a variable only if it is set.

The intruder can also create new messages and inject them into the network. So, the model can populate each constituent part of a message with some known information (between the states $I14$ and $I16$). While creating a new message, a local variable $data2$ can be set to any variable in the intruder's knowledge base. Then, it can be shifted left for $Length1$ or $Length2$ times depending on the length of the content to be appended to the message.

Our intruder model differs from the one in [17] and [18] by having reduced number of transitions depending on the reduced number of variables and using guarded transitions for the states used to improve its knowledge base.

V. VERIFICATION OF THE NEUMAN-STUBBLEBINE PROTOCOL

Timed automata model of a system can be verified using UPPAAL verification engine that uses a subset of CTL specifications. The tool checks these specifications by performing reachability analysis on the state space of the model. It has the advantage of generating a diagnostic trace that explains why a property is (or is not) satisfied.

In this case study, we aim to verify the correctness of an authentication protocol based on the security goals for a protocol. Two high level goals for an authentication protocol are listed as follows in [24]:

- *Authentication:* For each principal, after the successful run of the protocol, it should be assured that it is talking to the principal in its mind.
- *Key establishment:* A secret key becomes available to the principals, for subsequent cryptographic use.

We proposed to analyze the possible attacks for the Neuman-Stubblebine authentication protocol by writing specifications derived from these authentication goals. In order to examine the protocol goals given above, the *correspondence* and the *secrecy* properties should be verified. *Correspondence* means that the execution of different principals in an authentication protocol proceeds in a lock-stepped fashion. While the authenticating principal finishes its part of the protocol, the authenticated principal must have been present and participated in its part of the protocol. And, *secrecy* property specifies that a distributed session key cannot be discovered by the intruder.

In the analysis of these goals, if an attack is found on a protocol, it is inferred that the protocol is incorrect since it does not satisfy the properties that it is intended for.

This section gives the specifications and the corresponding UPPAAL queries that we used to check whether our protocol model satisfies these properties. Table 1 gives the verification results of these queries.

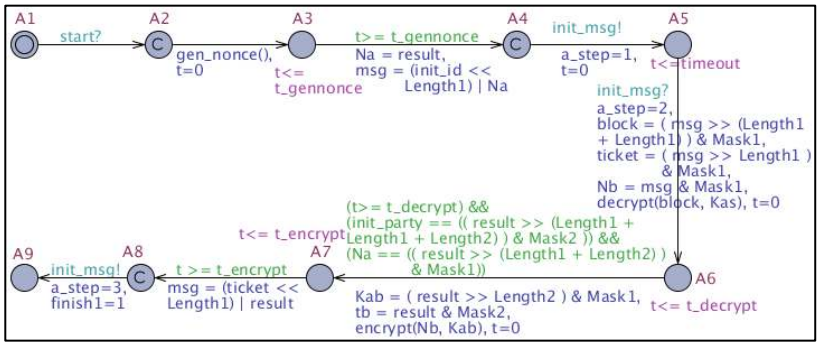

Figure 2. The initiator automaton

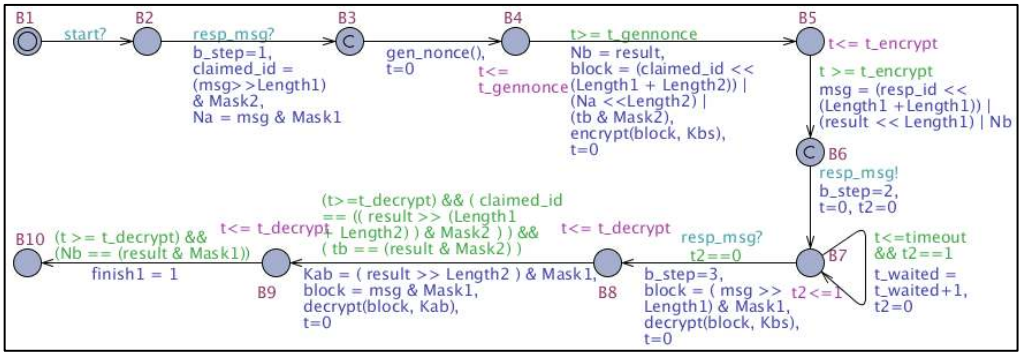

Figure 3. The responder automaton

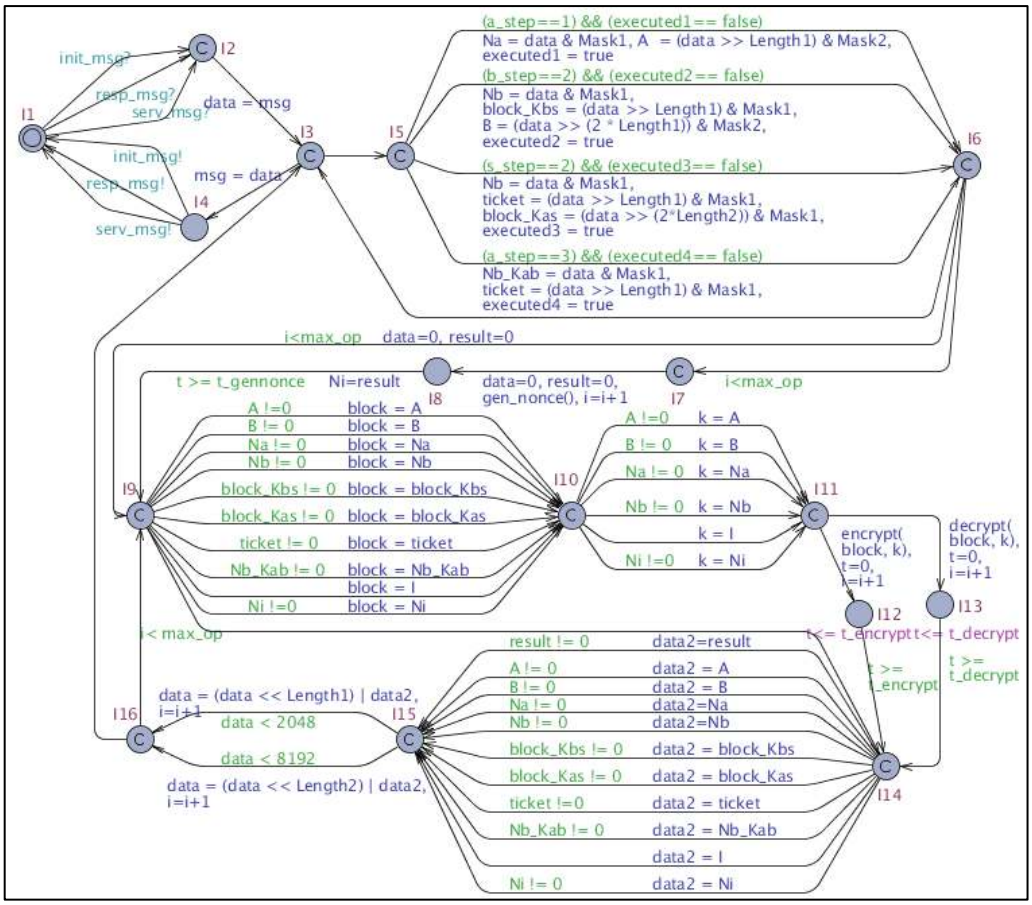

Figure 4. The intruder automaton

Query1: *Is such a state reachable where the responder finished but the initiator has not finished the initial protocol execution?*

$E <> Responder.finish1$ && (! $Initiator.finish1$)

Query 1 is related to the correspondence property. Here, we use the fact that this property is not satisfied when the responder finishes the protocol execution although the initiator has not executed its part. In such a situation, we can say that an intruder has sent fake messages to the responder to finish its protocol execution and attacked to the protocol.

However, this property is satisfied. It means that the intruder caused the responder to finish its run by sending fake messages and we have found an attack on the protocol. When we examined the diagnostic trace, we realized that this is the type flaw attack [25] given below. The intruder (see Fig. 4) extracts the information in messages 1 and 2 in the transitions from $I5$ to $I6$, and learns N_a, N_b, and $\{A, N_a, t_b\}_{K_{bs}}(block\_Kbs)$. It skips the protocol message 3. To create a fake message, it takes $block\_Kbs$ as $data2$ between the states $I14$, $I15$ and $I16$. Then, it selects N_b as $param1$ and N_a as $param2$, encrypts N_b with N_a. It composes this encrypted block with $block\_Kbs$ and sends it to the responder as the protocol message 4. In this attack, B accepts the nonce N_a as the key K_{ab}.

1. $I(A) \rightarrow B: A, N_a$
2. $B \rightarrow I(S): B, \{A, N_a, t_b\}_{K_{bs}}, N_b$
3. $omitted$
4. $I(A) \rightarrow B: \{A, N_a, t_b\}_{K_{bs}}, \{N_b\}_{N_a}$

As it is seen, a *type flaw* (substitution of a different type of message field) attack can be easily found by model checking with timed automata.

The next query is related to the *key distribution* that requires the new session key distributed by the server at most be known by the principals it is intended for.

Query2: *The execution of the protocol run leads to the fact that, the secret key acquired by the initiator is same with the secret key distributed to the responder which is also same with the key generated by the server.*

$(Responder.finish1$ && $Initiator.finish1) -\rightarrow$
$(Initiator.Kab == Responder.Kab == Server.Kab)$

The property is satisfied, that means all the executions where both the initiator and responder finished the initial part lead to the equivalence of Kab s owned by them.

In Query3, we test the *secrecy* property checking whether the intruder can learn the secret key. The check is performed using the $data2$ variable in order to include all the decomposed pieces of the messages and their encryptions or decryptions. This property is not satisfied, meaning that if the responder uses the key generated by the server (in the execution of a normal run); this secret key cannot be obtained by the intruder.

Query3: *Is such a state reachable where the secret key distributed to the principals is learned by the intruder?*

$E <> (Responder.finish1$ && $Initiator.finish1)$
&& $(Responder.Kab == Server.Kab ==$
$Intruder.data2)$

In the next query, our aim is to find out whether we can use the timing information to analyze the attacks on a protocol. The timeout intervals which are the time periods that a principal waits for a message can be used for this purpose. If a message comes earlier than the required time to prepare a message (depending on the encryption and the decryption times), then we can say that the principal has received a fake message [17][18].

The timeout intervals can be examined for both the initiator and the responder. To demonstrate the detection of an attack, here we give the query that examines the timeout for the responder. In a normal run, (assuming the time to create or read a message is negligible), the timeout for the responder is: $Responder.timeout \geq Server.t\_decrypt + 2 \times Server.t\_encrypt + Initiator.t\_decrypt + Initiator.t\_encrypt$ However, in a flawed run, the message can be received in a shorter time: $Responder.timeout \geq t\_encrypt$

To measure the time that the message is received, we use a local variable t_waited which is incremented at each time unit the responder waits (see Fig. 3, state $B7$). Query 4 is used to test for a possible attack using the fact that if the message comes earlier than the required time, then we can say that there is an attack on the protocol.

Query 4: Is such a state reachable where the responder has finished the protocol execution but the message has been received in a shorter time than the required time for the correct protocol execution?

$E <> Responder.finish1$ && $(Responder.t\_waited <$
$(Server.t\_decrypt + (2 \times Server.t\_encrypt) +$
$Initiator.t\_decrypt+Initiator.t\_encrypt)$

This property is satisfied and when we examined the diagnostic trace, we saw that it is the execution of the attack we have found in Query1. Hence, we infer that we can find the possible attacks on a protocol by examining the quantitative timing information and the flow of the protocol.

TABLE I. VERIFICATION RESULTS

No	States Stored	States Explored	Real Time	User Time	System Time	Satisfied
1	361610	428228	5.774s	5.424s	0.128s	Yes
2	1600933	3107116	28.021s	27.558s	0.408s	Yes
3	1550938	3057121	27.028s	26.598s	0.332s	No
4	361610	428228	5.584s	5.448s	0.088s	Yes

The queries of our case study are executed on Ubuntu 9.04 Operating System with Intel Core 2 Duo $P7350$, 2.00 GHz processor and $4GB$ of RAM. We used stand-alone command line verifier which is more appropriate for large verification tasks, with default configurations.

After the verification of the initial authentication part, we modeled the subsequent authentication in order to verify these parts together and also analyze the key expiration time. Consequently, the initiator, responder and the intruder automata are extended for this model which is not included in this paper. However, while executing our queries we have come up with state space explosion problem [23] caused by usage of large amount of memory. Hence, we could not obtain appreciable results.

Note that the subsequent part is exposed to a parallel session attack which cannot be detected by an automata model having one automaton for the initiator and one automaton for the responder allowing them to execute just one protocol run at a time. Because, the parallel session attack occurs when two protocol runs are executed concurrently and messages from one run are used to form fake messages in another run. The analysis of the combined Neuman-Stubblebine initial and subsequent authentication is left as future work because of state space explosion and the problem with parallel session attack.

VI. CONCLUSIONS & FUTURE WORK

Timed automata model for an authentication protocol can be used to examine the predefined goals of a protocol. Model checking of Neuman-Stubblebine initial authentication protocol with timed automata is able to find the type flaw attack using Dolev-Yao intruder model. In addition, this attack can also be detected by using the quantitative time information of the protocol.

The model can be further improved so that it can detect parallel session attacks that need the parallel execution of more than one protocol runs. Some work should be devoted to overcome the state space explosion that occurs for large models such as the verification of both initial and subsequent parts of the protocol.

In this paper, we concentrated on the verification of security protocols using timed automata formalism. The study can also be extended with a broad comparison of all of the security protocol verification methods which needs a deeper study on the other formalisms as well.

REFERENCES

[1] A. Cerone and A. Maggiolio-Schettini, "Time-based expressivity of timed petri nets for system specification", Elsevier Science Publishers Ltd., 1999, Theor. Comput. Sci, vol. 216, pp. 1-53.

[2] R. Alur, C. Courcoubetis and D. Dill, "Model-checking in dense real-time", Academic Press, Inc., 1993, Information and Computation, Vol 104, pp. 2-34.

[3] G. M. Reed and A. W. Roscoe, "A timed model for Communicating Sequential Processes", Elsevier Science Publishers Ltd., 1988, Theor. Comput. Sci., vol 58, pp. 249-261.

[4] R. Alur and D. Dill, "A theory of timed automata", Springer, 1994, Theoretical Computer Science, vol. 126, pp. 183-235.

[5] E.M. Clarke, O. Grumberg and D. A. Peled, Model Checking, Springer, 1999.

[6] B. C. Neuman and S. G. Stubblebine, "A note on the use of timestamps as nonces", ACM, 1993, SIGOPS Oper. Syst. Rev., vol. 27, pp. 10-14.

[7] G. Behrmann, A. David, K. Larsen, "A tutorial on UPPAAL", Springer, 2004, LNCS, Formal Methods for the Design of Real-Time Systems, vol 3185, pp. 200-236.

[8] H. Lönn, P. Pettersson, "Formal werification of a TDMA protocol start-up mechanism", IEEE Computer Society, 1997, In Pacific Rim International Symp. on Fault-Tolerant Systems, pp. 235-242.

[9] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi, "Automated analysis of an audio control protocol using UPPAAL", 2002, Journal of Logic and Algebraic Prog., vol. 52-53, pp 163-181

[10] K. Havelund, K. G. Larsen and A. Skou, "Formal verification of a power controller using the real-time model checker". Springer, 1999, Proc. of the 5th International AMAST Workshop on Real-Time and Probabilistic Systems, pp. 277-298.

[11] Bowman H., Faconti G., Katoen J-P, Latella D. and Massink M., "Automatic verification of a lip synchronisation algorithm using Uppaal", In Proc. of the 3rd Int. Workshop on Formal Methods for Industrial Critical Systems, 1998, pp. 97-124.

[12] G. Jakubowska, W. Penczek and M. Srebrny,"Verifying security protocols with timestamps via translation to timed automata", Proc. of the Int. Workshop on Concur., Spec. and Prog., 2005, Warsaw University Press, pp. 100–115 (2005)

[13] G. Jakubowska and W. Penczek, "Modelling and checking timed authentication of security protocols", IOS Press, 2008, Fundamenta Informatica, vol 79, pp. 363-378.

[14] S. Yovine, "KRONOS: a verification tool for real-time systems", Springer, 1997, International Journal on Software Tools for Technology Transfer, vol 1, pp. 123-133.

[15] P. Dembinski et. al, "VerICS: a tool for verifying timed automata and estelle specifications", TACAS 2003, Springer, 2003, LNCS, vol. 2619, pp. 278–283.

[16] M. Benerecetti and N. Cuomo, "TPMC: A model checker for time sensitive security protocols", Journal of Computers, 2009, vol. 4, pp. 366-377.

[17] R. Corin, S. Etalle, P. H. Hartel and A. Mader, "Timed model checking of security protocols", Proc. of the 2004 ACM workshop on Formal Methods in Security Engineering, pp. 23-32.

[18] R. Corin, S. Etalle, P. H. Hartel and A. Mader , "Timed analysis of security protocols", IOS Press, Journal of Computer Security, vol. 15, Dec 2007, pp. 619-645.

[19] R. Alur, C. Courcoubetis and D. Dill, "Model-checking for real-time systems", . In Proc. of the 5th IEEE Symp. on Logic in Comp. Science. IEEE, 1990.

[20] K. G. Larsen, P. Pettersson, and W. Yi, "Compositional and symbolic model-checking of real-time systems", RTSS '95: Proc. of the 16th IEEE Real-Time Systems Symp., 1995, pp. 76-87.

[21] J. Bengtsson, and W. Yi, "Timed automata: Semantics, algorithms and tools", Springer, 2004, LNCS, vol. 3098, pp. 87-124.

[22] D. Dolev and A. C. Yao, "On the security of public key protocols". IEEE Trans. on Information Theory, 29(2), pp. 198-208. 1983.

[23] R. Pelánek, "Fighting state space explosion: review and evaluation", Formal Methods for Industrial Critical Systems, FMICS 2008, Springer, 2009, LNCS, pp. 37-52.

[24] T. Y. C. Woo and S. S. Lam, "Design, verification and implementation of an authentication protocol", 1994, Proc of Int. Conf. on Network Protocols, pp 81-90.

[25] J. Clark and J. Jacob, "A survey of authentication protocol literature" Version 1.0, 1997.