Verifying Parameterized Timed Security Protocols

Li Li[1], Jun Sun[2], Yang Liu[3], and Jin Song Dong[1]

[1] National University of Singapore
[2] Singapore University of Technology and Design
[3] Nanyang Technological University

Abstract. Quantitative timing is often explicitly used in systems for better security, e.g., the credentials for automatic website logon often has limited lifetime. Verifying timing relevant security protocols in these systems is very challenging as timing adds another dimension of complexity compared with the untimed protocol verification. In our previous work, we proposed an approach to check the correctness of the timed authentication in security protocols with fixed timing constraints. However, a more difficult question persists, i.e., given a particular protocol design, whether the protocol has security flaws in its design or it can be configured secure with proper parameter values? In this work, we answer this question by proposing a parameterized verification framework, where the quantitative parameters in the protocols can be intuitively specified as well as automatically analyzed. Given a security protocol, our verification algorithm either produces the secure constraints of the parameters, or constructs an attack that works for any parameter values. The correctness of our algorithm is formally proved. We implement our method into a tool called PTAuth and evaluate it with several security protocols. Using PTAuth, we have successfully found a timing attack in Kerberos V which is unreported before.

1 Introduction

Time could be a powerful tool in designing security protocols. For instance, distance bounding protocols rely heavily on time; session keys with limited lifetime are extensively used in practice to achieve better security. However, designing timed security protocols is more challenging than designing untimed ones because timing adds a range of attacking surface, e.g., the adversary might be able to extend the session key without proper authorization. Hence, it is important to have a formal verification framework to analyze the timed security protocols. In our previous work [20], we developed a verification algorithm to analyze whether a given protocol with fixed timing constraints is secure or not. In this work, we answer a more difficult question, i.e., given a security protocol with configurable parameters for the timing constraints, are there any parameters which could guarantee security and what are they? Having an approach to answer the question is useful in a number of ways. Firstly, it can analyze, at once, all instances

of the security protocols with different parameter values. Secondly, it allows the protocol designer to gain precise knowledge on the secure configuration of the parameters so as to choose the best values (e.g., in terms of minimizing the protocol execution time).

In general, parameterized timing constraints are necessary in various scenarios. First of all, they can be used to capture the general design of the protocols. For instance, since the lifetime of credentials are often related to the runtime information like network latency, it is best to keep them parameterized so that we can systematically find out their secure relations. Furthermore, parameterized timing constraints are necessary to model the properties of some special cryptographic primitives. For example, weak cryptographic functions, which are breakable by consuming extra time, may be used in the sensor networks for higher computing performance and lower power consumption. Since breaking different weak functions requires different the attack time, in order to guarantee the correctness of the protocols in these sensor networks, we need to parameterize the attack time and compute the secure configuration accordingly. Moreover, agencies often give suggestions on key crypto-period for cryptographic key management [4], so parameterized timing constraints can be used to model long term protocols.

Nevertheless, this is a highly non-trivial task. The challenges for designing timed protocol and providing proper parameter configuration are illustrated as follows. First, in the setting of timed authentication over the Internet, given the network is completely exposed to the adversary, we need to formally prove that the critical information cannot be leaked and the protocol works as intended under arbitrary attacking behaviors from the network. Second, timestamps are continuous values extracted from clocks to ensure the validity of messages and credentials. Analyzing the continuous timing constraints adds another dimension of complexity. Third, a protocol design might contain multiple timing parameters, e.g., the network latency and the session key lifetime, which could affect security of the system. Manually reasoning the least constrained and yet correct configuration for the parameters in complex protocols is extremely hard and error-prone. As a consequence, automatic analysis technique is needed for proving the correctness of the protocol and computing the parameter configurations.

Contributions. Our contributions in this work are summarized as follows. (1) We propose an intuitive method to specify parameterized timed protocols in Section 3 by extending our previous work [20] with parameterized timing constraint, secrecy query, etc. (2) Based on the specification, protocols can be verified efficiently for an unbounded number of protocol sessions in our framework as shown in Section 4. Generally, in this work, we specify the adversary's capabilities in the security protocols as a set of Horn logic rules with parameterized timing constraints. Then, we compose these rules repeatedly until a fixed-point is reached, so that we can check the intended security properties against them and compute the largest parameter configurations. The parameter configuration is represented by succinct constraints of the parameters. When the protocol could be secure with the right parameter values, our approach outputs a set of constraints on

Type	Expression	
Message(m)	$f(m_1, m_2, ..., m_n)$	(function)
	$a[]$	(name)
	$[n]$	(nonce)
	v	(variable)
	t	(timestamp)
Parameter(p)	$\S p$	(parameter)
Constraint(B)	$\mathcal{C}(t_1, t_2, \ldots, t_n, \S p_1, \S p_2, \ldots, \S p_m)$	(timing constraint)
Configuration(L)	$\mathcal{C}(\S p_1, \S p_2, \ldots, \S p_m)$	(parameter configuration)
Event(e)	$know(m, t)$	(knowledge)
	$new([n], op[], \langle m_1, m_2, \ldots, m_n \rangle)$	(nonce generation)
	$init(m_1, m_2, \ldots, m_n)$	(init)
	$accept(m_1, m_2, \ldots, m_n)$	(accept)
	$leak(m)$	(leak)
Rule(R)	$[G]e_1, e_2, \ldots, e_n \multimap [\ B\ ] \mapsto e$	(rule)
	$e \leftarrow\!\!\multimap [\ B\ ] \vdash e_1, e_2, \ldots, e_n$	(query)

Table 1. Syntax Hierarchy Structure

the parameters that are necessary for security. Otherwise, an attack is generated, which would work for any parameter values. We formally prove the correctness of our algorithm. (3) We implement our method as a tool named PTAuth. In order to handle the parameters in the timing constraints, we utilize the Parma Polyhedra Library (PPL) [3] in our tool to represent the relations between timestamps and parameters. We evaluate our approach with several security protocols in Section 5. During the experiment, we found a timing attack in the official document of Kerberos V [27] that has never been reported before.

2 Running Example

We use the Wide Mouthed Frog (WMF) [8] protocol as a running example to illustrate how our approach works. WMF is designed for exchanging timely fresh session keys, ensuring that the key is generated by the protocol initiator within a short time when the protocol responder accepts it.

Syntax Hierarchy. Before describing the WMF protocol, we introduce the syntax for representing the messages first as shown in Table 1. *Messages* could be defined as *functions*, *names*, *nonces*, *variables* or *timestamps*. *Functions* can be applied to a sequence of *messages*; *names* are globally shared constants; *nonces* are freshly generated random values in sessions; *variables* are memory spaces for holding *messages*; and *timestamps* are values extracted from the global clock during the protocol execution. In addition, we introduce *parameters* to parameterize the timing constraints. The constraint function $\mathcal{C}(\mathbb{X})$ applies succinct constraints to $\mathbb{X}$, where $\mathbb{X}$ is a set of timestamps and parameters. Each succinct

constraint can be written in a general form of $l(t_1, \ldots, t_n, \S p_1, \ldots, \S p_m) \sim 0$, where $\sim \in \{<, \leq\}$ and l is a linear function. In the following paper, the symmetric encryption function is denoted as $enc_s(m, k)$, where m is the encrypted message and k is the encryption key. Furthermore, all the messages transmitted in WMF is encrypted by the shared key represented as $sk(u)$, which is only known between the user u and the server. For simplicity, the concatenation function $tuple_n(m_1, m_2, \ldots, m_n)$ is written as $\langle m_1, m_2, \ldots, m_n \rangle$ (or simply $m_1, m_2, \ldots, m_n$ when no ambiguity is introduced).

Events are constructed by attaching predicates to the message sequences. In our framework, we have five different predicates: (1) the knowledge event $know(m, t)$ means that the adversary knows the message m at the time t; (2) the nonce generation event $new([n], op[], \langle m_1, \ldots, m_n \rangle)$ means that a nonce $[n]$ is generated in the operation $op[]$ by a legitimate protocol participant with knowledge of $\langle m_1, \ldots, m_n \rangle$; (3) the event $init(m_1, \ldots, m_n)$ stands for the protocol initialization by a legitimate protocol participant with knowledge of $m_1, \ldots, m_n$; (4) similarly, the event $accept(m_1, \ldots, m_n)$ stands for the protocol acceptance by a legitimate protocol participant with knowledge of $m_1, \ldots, m_n$; (5) the event $leak(m)$ is introduced to check the leakage of the secret message m that violates the secrecy property, as shown in the example later.

Wide Mouthed Frog. The WMF protocol is a key exchange protocol consisting of three participants, i.e., the initiator *Alice*, the responder *Bob* and the server. It has the following five steps.

$$
\begin{array}{lll}
(1)\ \textit{Alice} \text{ engages} & : & new([k], alice\_gen[], \langle A[], B[], t_A \rangle) \\
 & & , init_A(A[], B[], [k], t_A) \\
(2)\ \textit{Alice} \rightarrow \textit{Server} & : & \langle A[], enc_s(\langle t_A, B[], [k] \rangle, sk(A[])) \rangle \\
(3)\ \textit{Server} \text{ checks} & : & t_S - t_A \leq \S p_a \\
\quad \textit{Server} \text{ engages} & : & init_S(A[], B[], [k], t_S) \\
(4)\ \textit{Server} \rightarrow \textit{Bob} & : & enc_s(\langle t_S, A[], [k] \rangle, sk(B[])) \\
(5)\ \textit{Bob} \text{ checks} & : & t_B - t_S \leq \S p_a \\
\quad \textit{Bob} \text{ engages} & : & accept(A[], B[], [k], t_B)
\end{array}
$$

First, *Alice* generates a fresh key $[k]$ at time t_A with the *new* event and engages an $init_A$ event to initiate the key exchange protocol with *Bob*. Second, *Alice* sends the fresh key with the current time t_A and *Bob*'s name to the server. Third, after receiving the request from *Alice*, the server checks the freshness of the timestamp t_A and accepts *Alice*'s request by engaging an $init_S$ event. Fourth, the server sends a new message to *Bob*, informing him that the server receives a request from *Alice* at time t_S to communicate with him using the key $[k]$. Fifth, *Bob* checks the timestamp and accepts the request from *Alice* if it is timely. The transmitted messages are encrypted under the users' shared keys.

Parameters. Whether or not WMF works relies on two crucial time parameters. The first parameter is the real network latency $\S p_d$ of the network, and the second one is the message delay $\S p_a$ allowed in the message freshness checking. $\S p_d$ is initially configured as $\S p_d > 0$ because the network latency should be positive. However, the exact value of $\S p_d$ depends on the network itself and thus cannot be

fixed in the protocol design. Parameter $\S p_a$ on the other hand might be related to $\S p_d$'s value, which should be answered by the verification. That is to say, the values of the parameters are better modeled as unknown parameters and we must be able to analyze the protocol without the concrete values of them. By introducing these two parameters, we want to make sure that the WMF protocol exchanges the secret session key successfully, and the correspondence between the request from Alice and the acceptance from Bob is timely. Hence, ideally a tool would automatically show us the secure configuration of $\S p_d$ and $\S p_a$. Because WMF has two message transmissions, we need to check whether $t_B - t_A \leq 2 * \S p_a$ is always satisfied.

3 Parameterized Timed Security Protocol Specification

In this section, we introduce how to model the parameterized timed security protocols. Generally, protocols as well as their underlying cryptography foundation are represented by a set of Horn logic rule variants [6] as shown in Table 1. They, denoted as $\mathbb{R}_{init}$, represent the capabilities of the adversary in the protocol.

Adversary Model. We assume that an active attacker exists in the network, extending from the Dolev-Yao model [15]. The attacker can intercept all communications, compute new messages, generate new nonces and send any message he obtained. For computation, he can use all the publicly available functions, e.g., encryption, decryption, concatenation. He can also ask the genuine protocol participants to take part in the protocol based on his needs. Comparing our attack model with the Dolev-Yao model, attacking the weak cryptographic functions and compromising legitimate protocol participant are allowed by consuming extra time, shown in Appendix C.

Rule Construction. Based on the adversary model described above, the interactions available to the adversary in the protocol can be represented by Horn logic rule variants guarded by timed checking conditions. Generally, every rule consists of a set of untimed guard conditions, several premise events, some timing constraints and one conclusion event as shown in Table 1. When the guard conditions, the premise events and the timing constraints in a rule are fulfilled, its conclusion event becomes available to the adversary. We remove the brackets if the rule has no guard condition. For instance, since the symmetric encryption and decryption functions are publicly available in WMF, these capabilities of the adversary can be represented by the following two rules.

$$know(m, t_1), know(k, t_2) \dashv\!\lfloor\, t_1, t_2 \leq t \,\rfloor\!\vdash know(enc_s(m, k), t) \quad (1)$$

$$know(enc_s(m, k), t_1), know(k, t_2) \dashv\!\lfloor\, t_1, t_2 \leq t \,\rfloor\!\vdash know(m, t) \quad (2)$$

The rule (1) means that given a message m and a key k, the adversary can compute its encryption $enc_s(m, k)$, and the encryption can only be known after the message and the key are obtained. Similarly, the rule (2) shows the decryption capability of the adversary.

Furthermore, the adversary can register new accounts at the server, except for the existing ones of *Alice* and *Bob*. So, we have the following rule.

$$[c \neq A[] \wedge c \neq B[]]know(c, t_1) \dashv\!\lfloor\ t_1 \leq t\ \rfloor\!\mapsto know(sk(c), t) \tag{3}$$

For rules related to the protocol itself, they can be extracted from the protocol readily. For instance, the adversary can actively ask *Alice* to initiate the first step of the WMF protocol, so the messages in the second step can be intercepted from the network, which is shown by the rule (4). As Alice can initiate this protocol with any user at any time based on the adversary's needs, the constant $B[]$ is replaced with a variable R and $know(\langle R, t_A\rangle, t)$ is added to the premises of the rule, comparing with protocol description in Section 2.

$$\begin{aligned} know(\langle R, t_A\rangle,t), new([k], alice\_gen[], \langle A[], R, t_A\rangle), init_A(A[], R, [k], t_A) \\ \dashv\!\lfloor\ t \leq t_A\ \rfloor\!\mapsto know(\langle A[], enc_s(\langle t_A, R, [k]\rangle, sk(A[]))\rangle, t_A) \end{aligned} \tag{4}$$

Similarly, based on the server's behavior (the third and fourth steps in WMF), we can construct the rule (5) shown below. Since the server provides its service to all of its users, *Alice* and *Bob*'s names are replaced by variables. The network latency and the message delay are captured by the parameterized constraints.

$$\begin{aligned} know(\langle I, enc_s(\langle t_I, R, k\rangle, sk(I))\rangle, t), init_S(I, R, k, t_S) \\ \dashv\!\lfloor\ t_S - t \geq \S p_d \wedge t_S - t_I \leq \S p_a\ \rfloor\!\mapsto know(enc_s(\langle t_S, I, k\rangle, sk(R)), t_S) \end{aligned} \tag{5}$$

Finally, *Bob* accepts the protocol when he receives the message from the server, indicating that the initiator is *Alice* and the request is fresh.

$$\begin{aligned} know(enc_s(\langle t_S, A[], k\rangle, sk(B[])), t) \\ \dashv\!\lfloor\ t_B - t \geq \S p_d \wedge t_B - t_S \leq \S p_a\ \rfloor\!\mapsto accept(A[], B[], k, t_B) \end{aligned} \tag{6}$$

Authentication Query. Similar to our previous work [20], verifying the timely authentication is allowed in our framework. The timely authentication not only asks for the proper correspondence between the init and accept events but also requires the satisfaction of the timing constraints, formalized as follows.

Definition 1. ***Timed Authentication.*** *In a timed protocol, timed authentication holds for an event accept with some events* $\{init_1, init_2, \ldots, init_n\}$ *agreed on the event arguments and the timing constraints* B*, if and only if for every occurrence of the event accept, all of the corresponding events* $\{init_1, init_2, \ldots, init_n\}$ *are engaged before, and their timestamps should always satisfy the timing constraints* B*. We denote the* timed authentication *query as accept* $\leftarrow\!\lfloor\ B\ \rfloor\!\vdash init_1, init_2, \ldots, init_n$*. In order to ensure the general timed authentication, the arguments encoded in the query events should only be variables and timestamps.*

In WMF, the authentication should be accepted by the responder R only if the request is made by the initiator I within $2 * \S p_a$. Thus, we have the following authentication query.

$$accept(I, R, k, k_R) \leftarrow\!\lfloor\ k_R - k_I \leq 2 * \S p_a\ \rfloor\!\vdash init_A(I, R, k, t_I) \tag{7}$$

Secrecy Query. In this work, we extend the verification algorithm developed in our previous work [20] with secrecy checking that can be relevant to timing. Secrecy checking is introduced with additional rules that lead to the leak events, representing the leakage of the secret information.

Definition 2. ***Secrecy.*** *In a security protocol, secrecy holds for a message m if the event $leak(m)$ is unreachable when "$new_1, new_2, \ldots, new_n, know(m,t) \; -[]\!\!\rightarrow leak(m)$" is added to $\mathbb{R}_{init}$, where $new_1, new_2, \ldots, new_n$ are the nonce generation events for all of nonces in m. Notice that all of the nonce generation events should have unique operation names so that they can be correctly identified.*

For instance, according to the WMF protocol, a secret session key $[k]$ is sent over the network. In order to check the secrecy property of $[k]$, we add the following rule to $\mathbb{R}_{init}$ and then check the reachability of the leak event.

$$new([k], alice\_gen[], \langle A[], B[], t_A \rangle), know([k], t) \; -[]\!\!\rightarrow leak([k]) \tag{8}$$

It means that if the session key $[k]$ generated by *Alice* for *Bob* can be known to the adversary, the secrecy property of the session key is invalid in WMF.

4 Verification Algorithm

Given a set of rules $\mathbb{R}$ and a parameter configuration L, we define $\alpha(\mathbb{R}, L) = \{H \; -[\; B \cap L \;]\!\!\rightarrow f | H \; -[\; B \;]\!\!\rightarrow f \in \mathbb{R}\}$, representing the rules under the parameter configuration L. Since the initial rules $\mathbb{R}_{init}$ can be extracted from the protocol as shown in Section 3, the satisfaction of an authentication query Q then depends on whether the adversary can actively guide the protocol to reach the *accept* event based on $\alpha(\mathbb{R}_{init}, L)$ without engaging the corresponding *init* events in Q or satisfying the timing constraints. Similarly, the verification of the secrecy query needs to check that the *leak* event is unreachable based on $\alpha(\mathbb{R}_{init}, L)$. In this section, we focus on computing the largest parameter configuration that ensures the correctness of the desired authentication and secrecy properties.

Given any parameter configuration L, in order to determine whether a query Q is satisfied by $\alpha(\mathbb{R}_{init}, L)$, we can adapt the verification algorithm in [20]. However, there might be infinitely many possible parameter configurations. Thus, in this work, we develop an approach to handle the parameters symbolically. Specifically, the verification is divided into two sequential phases: the rule basis construction phase and the query searching phase. In the rule base construction phase, we generate new rules by composing two rules (through unifying the conclusion of the first rule and the premise of the second rule). Our verification algorithm uses this method repeatedly to generate new rules until a fixed-point is reached. This fixed-point is called the *rule basis* if it exists. Subsequently, in the query searching phase, the query is checked against the *rule basis* to find counter examples. Generally, we need to check the event correspondence as well as the parameterized timing constraints, the verification either proves the correctness of the protocol by providing the secure configuration of the parameters

(represented as succinct constraints), or reports attacks because no parameter configuration can be found. Since the verification for security protocol is generally undecidable [12], our algorithm cannot guarantee termination. However, as shown in Section 5, our algorithm can terminate on most of the evaluated security protocols. Additionally, limiting the number of protocol sessions is allowed in our framework which would guarantee the termination of our algorithm.

Rule Basis Construction. Before constructing the rule basis, we need to introduce some basic concepts first:

– If σ is a substitution for both events e_1 and e_2 such that $\sigma e_1 = \sigma e_2$, we say e_1 and e_2 are unifiable and σ is an unifier for e_1 and e_2. If e_1 and e_2 are unifiable, the most general unifier for e_1 and e_2 is an unifier σ such that for all unifiers σ' of e_1 and e_2 there exists a substitution σ'' such that $\sigma' = \sigma''\sigma$.
– Given two rules $R = H \dashv\!\![\, B \,]\!\!\mapsto e$ and $R' = H' \dashv\!\![\, B' \,]\!\!\mapsto e'$, if e and $e_0 \in H'$ can be unified, their composition is denoted as $R \circ_{e_0} R' = \sigma(H \cup (H' - \{e_0\})) \dashv\!\![\, \sigma(B \cap B') \,]\!\!\mapsto \sigma e'$, where σ is the most general unifier of f and f_0.
– Additionally, given the above two rules R and R', we define R implies R' denoted as $R \Rightarrow R'$ when $\exists \sigma, \sigma e = e' \wedge \sigma H \subseteq H' \wedge B' \subseteq \sigma B$.

We construct the rule basis $\beta(\mathbb{R}_{init})$ based on the initial rules $\mathbb{R}_{init}$. Firstly, we define $\mathbb{R}_v$ as follows, representing the minimal closure of the initial rules $\mathbb{R}_{init}$. (1) $\forall R \in \mathbb{R}_{init}, \exists R' \in \mathbb{R}_v, R' \Rightarrow R$, which means that every initial rule is implied by a rule in $\mathbb{R}_v$. (2) $\forall R, R' \in \mathbb{R}_v, R \not\Rightarrow R'$, which means that no duplicated rule exists in $\mathbb{R}_v$. (3) $\forall R, R' \in \mathbb{R}_v$ and $R = H \dashv\!\![\, B \,]\!\!\mapsto e$, if $\forall e' \in H, e' \in \mathbb{V}$ and $\exists e_0 \notin \mathbb{V}, S \circ_{e_0} S'$ is defined, then $\exists S'' \in \mathbb{R}_v, S'' \Rightarrow R \circ_{e_0} R'$, where $\mathbb{V}$ is a set of events that can be provided by the adversary. In this work, $\mathbb{V}$ is the *init* events, the *new* events and the $know(x, t)$ event where x is a variable. The third rule means that for any two rules in $\mathbb{R}_v$, if all premises of one rule are trivially satisfiable and their composition exists, their composition is implied by a rule in $\mathbb{R}_v$. Based on $\mathbb{R}_v$, we can calculate the rule basis as follows.

$$\beta(\mathbb{R}_{init}) = \{R \mid R = H \dashv\!\![\, B \,]\!\!\mapsto e \in \mathbb{R}_v \wedge \forall e' \in H : e' \in \mathbb{V}\}$$

Theorem 1. *For any rule R in the form of $H \dashv\!\![\, B \,]\!\!\mapsto e$ where $\forall e' \in H : e' \in \mathbb{V}$, R is derivable from $\alpha(\mathbb{R}_{init}, L)$ if and only if R is derivable from $\alpha(\beta(\mathbb{R}_{init}), L)$.*

Due to the limitation of space, the proof is presented in Appendix B.

Query Searching. A rule is a contradiction to the authentication query if and only if its conclusion event is an *accept* event, while it does not require all the *init* events or it has looser timing constraints comparing with those in the query. Otherwise, if the rule's conclusion event is an *accept* event while this rule is not a contradiction to the authentication query, then it is an obedience. Similarly, a rule is a contradiction to the secrecy query when the *leak* event is reachable.

Definition 3. ***Authentication Contradiction and Obedience.*** *A rule $R = H \dashv\!\![\, B \,]\!\!\mapsto e$ is a contradiction to the authentication query $Q_a = e' \leftarrow\!\![\, B' \,]\!\!\vdash H'$ denoted as $Q_a \nvDash R$ if and only if $B \neq \emptyset$, e and e' are unifiable with the most*

Algorithm 1: Parameter Configuration Computation

Input : $\beta(\mathbb{R}_{init})$, L_0 - the rule basis and the initial configuration
Input : $\mathbb{Q}_A, \mathbb{Q}_S$ - the authentication and secrecy queries
Output: $\mathbb{L}$ - the set of parameter configurations

1 **Algorithm**
2 $\quad \mathbb{L} = \{L_0\}$;
3 $\quad$ **for** $Q \in \mathbb{Q}_A \cup \mathbb{Q}_S, L \in \mathbb{L}, R = H \dashv\!\lfloor\ B\ \rfloor\!\!\rightarrow f \in \alpha(\beta(\mathbb{R}_{init}), L), Q \nvdash R$ **do**
4 $\quad\quad \mathbb{L} = \mathbb{L} - \{L\}$;
5 $\quad\quad$ **for** $L' : B \cap L' = \emptyset \vee Q \vdash H \dashv\!\lfloor\ B \cap L'\ \rfloor\!\!\rightarrow f$ **do** $\mathbb{L} = \mathbb{L} \cup \{L \cap L'\}$;
6 $\quad$ **for** $L \in \mathbb{L}, Q \in \mathbb{Q}_A$ **do**
7 $\quad\quad$ **if** *cannot find* $R \in \alpha(\beta(\mathbb{R}_{init}), L), Q \vdash R$ **then** $\mathbb{L} = \mathbb{L} - \{L\}$;
8 $\quad$ **return** $\mathbb{L}$;

general unifier σ *and* $\forall\sigma', (\sigma'\sigma H' \not\subseteq \sigma H) \vee (\sigma B \not\subseteq \sigma'\sigma B')$. *On the other hand, it is an obedience denoted as* $Q_a \vdash R$ *if and only if* $B \neq \emptyset$, e *and* e' *are unifiable with the most general unifier* σ *and* $\exists\sigma', (\sigma'\sigma H' \subseteq \sigma H) \wedge (\sigma B \subseteq \sigma'\sigma B')$.

Definition 4. ***Secrecy Contradiction.*** *A rule* $R = H \dashv\!\lfloor\ B\ \rfloor\!\!\rightarrow e$ *is a contradiction to the secrecy query* Q_s *of message* m *denoted as* $Q_s \nvdash R$ *if and only if* $B \neq \emptyset \wedge e = leak(m) \wedge \forall e' \in H : e' \in \mathbb{V}$.

During the verification, our goal is to ensure that (1) no contradiction rule exists for all queries while (2) at least one obedience rule exists for every authentication query. Hence, given the authentication queries $\mathbb{Q}_A$ and the secrecy queries $\mathbb{Q}_S$, our goal is to compute the largest L that satisfies the following two conditions: (1) $\forall Q \in \mathbb{Q}_A \cup \mathbb{Q}_S, \{R | R \in \alpha(\beta(\mathbb{R}_{init}), L) \wedge Q \nvdash R\} = \emptyset$; (2) $\forall Q \in \mathbb{Q}_A, \{R | R \in \alpha(\beta(\mathbb{R}_{init}), L) \wedge Q \vdash R\} \neq \emptyset$. Algorithm 1 illustrates the computing process of the largest L. From line 3 to line 5, we compute the parameter configurations that remove the contradictions. From line 6 to line 7, we ensure that every authentication query has at least one obedience. In order to prove the correctness of our algorithm, we need to show that for any configuration L, a contradiction exists in $\alpha(\beta(\mathbb{R}_{init}), L)$ if and only if it exists in $\alpha(\mathbb{R}_{init}, L)$.

Theorem 2. ***Partial Correctness.*** *Let* Q *be the query and* $\mathcal{R}_{init}$ *be the initial rule set. There exists* R *derivable from* $\alpha(\mathcal{R}_{init}, L)$ *such that* $Q \nvdash R$ *if and only if there exists* $R' \in \alpha(\beta(\mathbb{R}_{init}), L)$ *such that* $Q \nvdash R'$.

Due to the limitation of space, the proof is available in the Appendix B.

Checking WMF. After checking the specification of WMF using the abovementioned algorithm, PTAuth claims an attack. The two key rules in $\beta(\mathbb{R}_{init})$ are shown below. The rule (9) represents the execution trace that the server transmits the key once from *Alice* to *Bob*. It is obedient to the query (7). However, the rule (10) is a contradiction to the query (7), because it has a weaker timing range ($t_B \leq t_A + 4 * \S p_a$) than that in the query ($t_B \leq t_A + 2 * \S p_a$). This

rule stands for the execution trace that the adversary sends the message from the server back to server twice and then forwards it to *Alice*. According to the rule (5), the timestamp in the message can be updated in this method. Hence, *Bob* would not notice that the message is actually delayed when he receives it. In order to remove the contradiction, we need to configure the parameters as either $\S p_a < \S p_d$ or $\S p_a \leq 0$. However, applying any one of these constraints to the initial configuration $0 < \S p_d$ leads to the removal of the rule (9), the only obedience rule in $\alpha(\beta(\mathbb{R}_{init}), L)$. Hence, PTAuth claims that an attack is found, which means that no parameter configuration would make the protocol work.

$$
\begin{aligned}
&know(t_A, t), \mathbf{new}([k], alice\_gen[], \langle A[], B[], t_A\rangle)\\
&, \mathbf{init_A}(A[], B[], [k], t_A), \mathbf{init_S}(A[], B[], [k], t_S)\\
&\quad \dashv[\, t \leq t_A, t_B \leq t_S + \S p_a \leq t_A + 2 * \S p_a,\\
&\qquad\qquad t_A + 2 * \S p_d \leq t_S + \S p_d \leq t_B,]\mapsto\\
&\qquad\qquad\qquad\qquad \mathbf{accept}(A[], B[], [k], t_B) \qquad (9)\\
&know(t_A, t), \mathbf{new}([k], alice\_gen[], \langle A[], B[], t_A\rangle)\\
&, \mathbf{init_A}(A[], B[], [k], t_A), \mathbf{init_S}(A[], B[], [k], t_{S1})\\
&, \mathbf{init_S}(B[], A[], [k], t_{S2}), \mathbf{init_S}(A[], B[], [k], t_{S3})\\
&\quad \dashv[\, t \leq t_A, t_B \leq t_{S3} + \S p_a \leq t_{S2} + 2 * \S p_a \leq t_{S1} + 3 * \S p_a \leq t_A + 4 * \S p_a,\\
&\qquad\qquad t_A + 4 * \S p_d \leq t_{S1} + 3 * \S p_d \leq t_{S2} + 2 * \S p_d \leq t_{S3} + \S p_d \leq t_B\,]\mapsto\\
&\qquad\qquad\qquad\qquad \mathbf{accept}(A[], B[], [k], t_B) \qquad (10)
\end{aligned}
$$

Corrected WMF. The WMF protocol can be fixed by inserting two different constants to the messages sent to and received from the server respectively, which breaks their symmetric structure. Using this method, the server can distinguish the messages that it sent out previously, and refuse to process them again. Our algorithm proves the correctness of this modified WMF protocol, and produces the timing constraints $0 < \S p_d \leq \S p_a$.

5 Evaluations

Based on our verification framework, we have implemented a tool named PTAuth. We encode PPL [3] in our tool to analyze the satisfaction of timing constraints. Meanwhile, in order to improve the performance, we implement an on-the-fly verification algorithm that updates the parameter configuration whenever a rule is generated. Hence, the verification process can terminate early if an attack can be found. We use PTAuth to check many security protocols as shown in Table 2. All the experiments shown in this section are conducted under Mac OS X 10.10.1 with 2.3 GHz Intel Core i5 and 16G 1333MHz DDR3. In the experiments, we have checked several timed protocols i.e., the WMF protocols [8,14], the Kerberos protocols [27], the distance bounding protocolse [7,10,28] and the CCITT protocols [11,2,8]. Additionally, we analyze the untimed protocols like the Needham-Schroeder series [26,21] and SKEME [18]. As can be seen, most of

Protocol	Parameterized	Bounded	$\sharp\mathcal{R}^a$	Result	Time
Wide Mouthed Frog [8]	Yes	No	40	Attack [22]	39ms
Wide Mouthed Frog (c) [14]	Yes	No	35	Secure	13ms
Kerberos V [27]	Yes	No	19370	Attack	23m5s
Kerberos V (c)	Yes	Yes	438664	Secure	2h41m
Auth Range [7,10]	Yes	No	21	Secure	10ms
Ultrasound Dist Bound [28]	Yes	No	50	Attack [29]	18ms
CCITT X.509 (1) [11]	No	No	45	Attack [2]	14ms
CCITT X.509 (1c) [2]	No	No	62	Secure	37ms
CCITT X.509 (3) [11]	No	No	127	Attack [8]	84ms
CCITT X.509 (3) BAN [8]	No	No	148	Secure	131ms
NS PK [26]	No	No	68	Attack [21]	30ms
NS PK Lowe [21]	No	No	61	Secure	28ms
SKEME [18]	No	No	127	Secure	466ms

Table 2. Experiment Results

[a] Rules generated in the verification.

the protocols can be verified or falsified by PTAuth quickly for an unbounded number of protocol sessions. Notice that the secure configuration is given based on the satisfaction of all of the queries, so we do not show the results for different queries separately in the table. The justification for the bounded verification of the corrected version of Kerberos V is presented later in this section. The PTAuth tool and the models shown in this section are available in [1]. Particularly, we have successfully found a new attack in Kerberos V [27] using PTAuth. In the following, we present the detailed findings in Kerberos V. Since Kerberos V is the latest version, we denote it as Kerberos for short unless otherwise indicated.

Kerberos Overview. Kerberos is a widely used security protocol for accessing services. For instance, Microsoft Window uses Kerberos as its default authentication method; many UNIX and UNIX-like operating systems include software for Kerberos authentication. Kerberos has a salient property such that its user can obtain accesses to a network service within a period of time using a single request. In general, this is achieved by granting an access ticket to the user, so that the user can subsequently use this ticket to authenticate himself to the server. Kerberos is complex because multiple ticket operations are supported simultaneously and many fields are optional, which are heavily relying on time. So, configuring Kerberos is hard and error-prone.

Kerberos consists of five types of entities: *User*, *Client*, *Kerberos Authentication Server* (KAS), *Ticket Granting Server* (TGS) and *Application Server* (AP). KAS and TGS together are also known as *Key Distribution Centre* (KDC). Specifically, *Users* usually are humans, and *Clients* represent their identities in the Kerberos network. KAS is the place where a *User* can initiate a logon session to the Kerberos network with a pre-registered *Client*. In return, KAS provides the *User* with (1) a *Ticket Granting Ticket* (TGT) and (2) an encrypted session

key as the authorization proof to access TGS. After TGS checks the authorization from KAS, TGS issues two similar credentials (1) a *Service Ticket* (ST) and (2) a new encrypted session key to the *User* as authorization proof to access AP. Then, the *User* can finally use them to retrieve the *Service* from AP. Additionally, both of the TGT and the ST can be postdated, validated and renewed by KDC when these operations are permitted in the Kerberos network.

Specification Highlights. Generally, by following the method described in Section 3, the specification for Kerberos itself can be extracted easily. In order to verify Kerberos comprehensively, we model several keys and timestamps (which could be optional) by following precisely its official document RFC 4120 [27].

- The user and the server are allowed to specify sub-session keys in the messages. When a sub-session key is specified, the message receiver must use it to transmit the next message rather than using the default session-key.
- Optional timestamps are allowed in the user requests and the tickets. In the following paper, fq, tq and rq denote the start-time, the end-time and the maximum renewable end-time requested by the users. Similarly, sp, ep and rp denote the start-time, the end-time and the maximum renewable end-time agreed by the servers. sp, ep and rp are encoded in the tickets, corresponding to fq, tq and rq respectively. An additional timestamp ap is encoded in the ticket to represent the initial authentication time of the ticket. Furthermore, cq represents the current-time when the request is made by the user, and cp stands for the current-time when the ticket is issued by the server. In Kerberos, fq, rq, sp and rp are optional. So the servers need to check their presence and construct replies accordingly.

In this work, two parameters are considered in Kerberos, i.e., the maximum lifetime $\S l$ and the maximum renewable lifetime $\S r$ of the tickets. Based on these parameters, the servers can only issue tickets whose lifetime and renewable lifetime are shorter than $\S l$ and $\S r$ respectively. Furthermore, five operations are modeled for the Kerberos servers as follows. (1) Postdated tickets can be generated for future usage. They are marked as invalid initially and they must be validated later. (2) Postdated tickets must be validated before usage. (3) Renewable tickets can be renewed before they expire. (4) Initial tickets are generated at KAS using user's client. (5) Sub-tickets are generated at TGS using existing tickets. Notice that the end-time ep of the sub-ticket should be no larger than the end-time of the existing ticket. Due to the space limit, we provide the complete model of Kerberos in Appendix A.

Queries. In order to specify the queries, we define three events as follows.

- When an initial ticket is generated at KAS, an $\mathbf{init_{auth}}(k, C, S, t)$ event is engaged, where k is the fresh session key, C is the client's name, S is the **target** server's name, and t is the beginning of the ticket's lifetime.
- Whenever an new ticket is generated at KAS or TGS, an $\mathbf{init_{gen}}(k, C, S, t)$ event is engaged. Its arguments have the same meaning as those in $init_{auth}$.

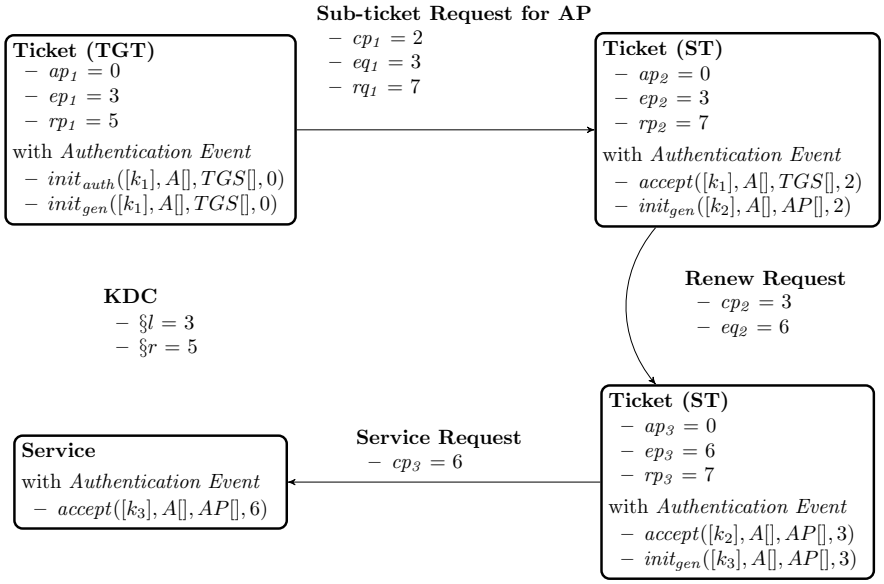

Fig. 1. Attack Found in Kerberos V

– Whenever an ticket is accepted by the server, an $\mathbf{accept}(k, C, S, t)$ event is engaged, where k is the agreed session key, C is the client's name, S is the **current** server's name, and t is the acceptance time.

In Kerberos, we need to ensure the correctness of two timed authentications. First, whenever a server accepts a ticket, the ticket should be indeed generated within $\S l$ time units using the same session key. Second, whenever a server accepts a ticket, the initial ticket should be indeed generated within $\S r$ time units.

$$accept(k, C, S, t) \leftarrow\!\!\lfloor\ t - t' \leq \S l\ \rfloor\!\!- init_{gen}(k, C, S, t') \tag{11}$$

$$accept(k, C, S, t) \leftarrow\!\!\lfloor\ t - t' \leq \S r\ \rfloor\!\!- init_{auth}(k', C, S', t') \tag{12}$$

Verification Results. For the termination of the verification, we need to initially configure the parameters as $\S r < n * \S l$, where n can be any integer larger than 1. The requirement for this constraint is justified as follows. Algorithm 1 updates parameter configuration at line 5 to eliminate the contradiction rules. Suppose we have a rule $init_{auth}(k, C, S, t') \dashv\!\lfloor\ t - t' \leq c * \S l\ \rfloor\!\!\mapsto accept(k, C, S, t)$ in the rule basis, where $c > 1$. This rule is a contradiction to the query (12) because $\S r$ is not necessarily larger than $c * \S l$. However, Algorithm 1 can add a new constraint $c * \S l \leq \S r$ to the existing configuration and then continue searching. Since we have infinitely many such rules in $\beta(\mathbb{R}_{init})$ with different values of c, the verification cannot terminate. Hence, in this work, we set the initial configuration as $\S r < 2 * \S l$ to avoid the non-termination. Notice that this initial configuration does not prevent us from finding attacks because it does not limit the number of sequential operations allowed in the Kerberos protocol.

After analyzing Kerberos using PTAuth, we have successfully found a security flaw in its specification document RFC 4120 [27]. The attack trace is depicted in Figure 1. Suppose the Kerberos is configured with $\S l = 3$ and $\S r = 5$[4], and a user Alice has already obtained a renewable ticket at time 0. Then, she can request for a sub-ticket of AP at time 2 that is renewable until time 7, satisfying $rq_1 - cp_1 \leq \S r$. Notice the new sub-ticket's end-time ep_2 cannot be larger than the end-time ep_1 of the existing ticket. Later, she renews the new sub-ticket before it expires and gets a ticket valid until time 6. Finally, she requests the service at time 6 and engages an event $accept([k_3], A[], AP[], 6)$. However, this accept event does not correspond to any $init_{auth}$ event satisfying Query (12), which leads to an attack. In fact, Alice can use this method to request sub-ticket for AP repeatedly so that she can have access to the service forever. Obviously, the server who made the authentication initially does not intend to do so. Fortunately, after checking the source code of Kerberos, we find that this flaw is prevented in its implementations [24,19]. An additional checking condition[5] has been inserted to regulate that the renewable lifetime in the sub-ticket should be smaller than the renewable lifetime in the existing ticket. We later confirmed with Kerberos team that this is an error in its specification document, which could have led to a security issue but has not done so in its current implementation.

Corrected Version. After adding the timing constraints on renewable lifetime between the base-ticket and the sub-ticket, the verification cannot terminate. This is caused by an infinite dependency trace formed by tickets, as we do not limit its length. Hence, we bound the number of tickets that can be generated during the verification, which in turn bounds the number of $init_{gen}$ events in the rule. In this work, we bound the ticket number to five. This is justified as we have five different methods to generate tickets in Kerberos: the servers can postdate, validate, renew tickets, generate initial tickets and issue sub-tickets. After bounding the ticket number that can be generated, our tool proves the correctness of Kerberos and produces the configuration $0 \leq \S l \leq \S r < 2 * \S l$.

6 Related Works

As mentioned, this work is related to our previous work [20]. In this work, we additionally introduce timing parameters, secrecy queries, etc. and enhance the computation capability of the timing constraint with PPL. Furthermore, we provide the algorithm to compute the least constrained secure configuration of parameters in this work. We successfully analyze several protocols including Kerberos V and find an attack in the Kerberos V specification [27] that is unreported before. The analyzing framework closest to ours was proposed by Delzanno and Ganty [14] which applies $MSR(\mathcal{L})$ to specify unbounded crypto protocols by combining first order multiset rewriting rules and linear constraints. According

[4] $\S l$ and $\S r$ are represented by symbols during the verification.

[5] For krb5-1.13 from MIT, the checking is located in the file src/kdc/kdc_util.c at line 1740 - 1741. We also checked other implementations, like heimdal-1.5.2.

to [14], the protocol specification is modified by explicitly encoding an additional timestamp, representing the initialization time, into some messages. Thus the attack can be found by comparing the original timestamps with the new one in the messages. However, it is unclear how to verify timed protocol in general using their approach. On the other hand, our approach can be applied to protocols without any protocol modification. Many tools for verifying protocols [6,13,23] are related. However, they are not designed for timed protocols.

Kerberos has been scrutinized over years using formal methods. In [5], Bella et al. analyzed Kerberos IV using the Isabelle theorem prover. They checked various secrecy and authentication properties and took time into consideration. However, Kerberos is largely simplified in their analysis and the specification method in their work is not as intuitive as ours. Later, Kerberos V has been analyzed by Mitchell et al. [25] using state exploration tool Murφ. They claimed that an attack is found in [17] when two servers exists. However, this attack is actually prevented in Kerberos's official specification document RFC 1510 [16], which is later superseded by RFC 4120 [27] analyzed in this paper. The biggest advantages of our method is that the verification is given for an unbounded number of sessions, which is not achievable previously with the state exploration approach. For the above literatures, they did not consider alternative options supported in the protocol that may accidentally introduce attacks as we do in this work. Similar to our work, Kerberos V has been analyzed in a theorem proving context by Butler et al. [9]. They took many features into consideration, i.e., the error messages, the encryption types and the cross-realm support. These features are not cover in this work since we focus on the timestamps and timing constraint checking. Meanwhile, our framework can provide intuitive modeling and automatic verifying, while Kerberos V is analyzed manually in [9].

7 Conclusions

In this work, we developed an automatic verification framework for timed parameterized security protocols. It can verify authentication properties as well as secrecy properties for an unbounded number of protocol sessions. We have implemented our approach into a tool named PTAuth and used it to analyze a wide range of protocols shown in Section 5. In the experiments, we have found a timed attack in Kerberos V document that has never been reported before.

Since the problem of verifying security protocols is undecidable in general, we cannot guarantee the termination of our verification algorithm. When we use PTAuth to analyze the corrected version of Kerberos, PTAuth cannot terminate because of the infinite dependency chain of tickets. Hence, we have to bound the number of tickets generated in the protocol. However, in Kerberos, generating more tickets may not be helpful to break its security. Based on this observation, we want to detect and prune the non-terminable verification branches heuristically without affecting the final results in our future work. This could help us to verify large-sized and complex protocols that we cannot verify currently, as our verification algorithm only considers the general approach at present.

References

1. PTAuth extended paper, tool and experiment models. Available at `http://www.comp.nus.edu.sg/~li-li/r/time.html`.
2. M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996.
3. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the parma polyhedra library. In *SAS*, pages 213–229. Springer, 2002.
4. E. B. Barker, W. C. Barker, W. E. Burr, W. T. Polk, and M. E. Smid. SP 800-57. Recommendation for key management. Technical report, National Institute of Standards & Technology, 2007.
5. G. Bella and L. C. Paulson. Kerberos version 4: Inductive analysis of the secrecy goals. In *ESORICS*, pages 361–375. Springer, 1998.
6. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW*, pages 82–96. IEEE CS, 2001.
7. S. Brands and D. Chaum. Distance-bounding protocols (extended abstract). In *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 1993.
8. M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.
9. F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad. Formal analysis of kerberos 5. *Theor. Comput. Sci.*, 367:57–87, 2006.
10. S. Capkun and J.-P. Hubaux. Secure positioning in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2):221–232, 2006.
11. CCITT. The directory authentication framework - Version 7, 1987. Draft Recommendation X.509.
12. I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *CSFW*, pages 55–69. IEEE CS, 1999.
13. C. Cremers. The Scyther tool: Verification, falsification, and analysis of security protocols. In *CAV*, pages 414–418. Springer, 2008.
14. G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. In *TACAS*, pages 342–356. Springer, 2004.
15. D. Dolev and A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.
16. J. Kohl and B. C. Neuman. *The Kerberos Network Authentication Service (Version 5). Internet Request for Comments RFC-1510.* RFC Editor, 1993.
17. J. T. Kohl, B. C. Neuman, and T. Y. T'so. The evolution of the kerberos authentication system. In *Distributed Open Systems*, pages 78–94. IEEE CS, 1994.
18. H. Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. In *NDSS*, pages 114–127. IEEE CS, 1996.
19. LDAP Account Manager. Kerberos V implementation heimdal-1.5.2. `http://www.h5l.org`, 2014.
20. L. Li, J. Sun, Y. Liu, and J. S. Dong. Tauth: Verifying timed security protocols. In *ICFEM*, pages 300–315. Springer, 2014.
21. G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.
22. G. Lowe. A family of attacks upon authentication protocols. Technical report, Department of Mathematics and Computer Science, University of Leicester, 1997.

23. S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The Tamarin prover for the symbolic analysis of security protocols. In *CAV*, pages 696–701. Springer, 2013.
24. MIT. Kerberos V implementation krb5-1.13. `http://web.mit.edu/kerberos/`, 2014.
25. J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ;. In *S&P*, pages 141–151, 1997.
26. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.
27. C. Neuman, T. Yu, S. Hartman, and K. Raeburn. *The Kerberos Network Authentication Service (Version 5). RFC-4120.* RFC Editor, 2005.
28. N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *Workshop on Wireless Security*, pages 1–10. ACM, 2003.
29. S. Sedighpour, S. Capkun, S. Ganeriwal, and M. B. Srivastava. Implementation of attacks on ultrasonic ranging systems (demo). In *SenSys*, page 312. ACM, 2005.

Appendix A: The Kerberos Protocol

The execution trace of Kerberos is shown in Table 3. Based on the optional keys and timestamps presented in the user's request and ticket, different checking constraints are used by the server and different tickets thus are issued. In Table 4, we describe two kinds policies considered in the verification. The first type of policies grants permissions to server operations in the Kerberos network, e.g., whether renewing tickets and postdating tickets are allowed. The second type of policies specifies qualitative parameters as symbolic values, e.g., the maximum life-time and renewable life-time of tickets. Generally, the first type of policies gives branches to the protocol execution, while the second type of policies parameterizes the timing constraints checked in the protocol. We list all of the timing constraints checked in different protocol execution branches in Table 5.

Appendix B. Proofs

Lemma 1. *For any rule R in the form of $H -[\ \mathcal{B}\ ]\rightarrow e$ where $\forall e' \in H : e' \in \mathbb{V}$, R is derivable from $\mathbb{R}_{init}$ if and only if R is derivable from $\beta(\mathbb{R}_{init})$.*

Proof. This lemma has already been proved in [20].

Definition 5. ***Derivation Tree.*** *Let $\mathbb{R}$ be a set of closed rules and R be a closed rule, where a closed rule is a rule with its conclusion initiated by its premises. Let R be a rule in the form of $e_1, \ldots, e_n -[\ B\ ]\rightarrow e$. R can be derived from $\mathbb{R}$ if and only if there exists a finite derivation tree defined as*

1. *edges in the tree are labeled by events;*
2. *nodes are labeled by the rules in $\mathbb{R}$;*
3. *if a node labeled by R has incoming edges of $e'_1, \ldots, e'_n$, an outgoing edge of e', and the timestamps among these facts satisfy the timing constraints $\mathcal{B}'$, then $S \Rightarrow e'_1, \ldots, e'_n -[\ B'\ ]\rightarrow e'$;*
4. *the outgoing edge of the root is the event e;*
5. *the incoming edges of the leaves are $e_1, \ldots, e_n$.*

Additionally, if all the timing constraints in the derivation tree form B, then the timing constraints for R is $sim(B)$, where sim removes timestamps that are no longer used. We name this tree as the derivation tree of R based on $\mathbb{R}$.

Theorem 1. For any rule R in the form of $H -[\ B\ ]\rightarrow e$ where $\forall e' \in H : e' \in \mathbb{V}$, R is derivable from $\alpha(\mathbb{R}_{init}, L)$ if and only if R is derivable from $\alpha(\beta(\mathbb{R}_{init}), L)$.

Proof. Given a rule $R = H -[\ B\ ]\rightarrow e$, we define $\pi(R, L) = H -[\ B \cap L\ ]\rightarrow e$. Given a derivation tree T of R, we define $\Pi(T, L)$ as a derivation tree whose node label R is replaced with $\pi(R, L)$. According to Lemma 1, $R = H -[\ B\ ]\rightarrow e$ is derivable from $\mathbb{R}_{init}$ if and only if R is derivable from $\beta(\mathbb{R}_{init})$. It means that we can construct a derivation tree T of R based on $\mathbb{R}_{init}$ if and only if we can construct a derivation tree T' of R based on $\beta(\mathbb{R}_{init})$. After applying the configuration L to all of the labels of T, we have the following two conditions.

Phase 1 : Get TGT or ST using Shared Key (*User* $\leftrightarrow$ *Kerberos Authentication Server*)		
KRB_AS_REQ($U \rightarrow K$)	C, S, Q, $[n]$	;*Request body*
AS	if sp' presents then $init_{auth}([k_S], C, S, sp')$	;*Server authentication*
	else $init_{auth}([k_S], C, S, cp)$	;*Server authentication*
	if sp' presents then $init_{gen}([k_S], C, S, sp')$	;*Ticket generation*
	else $init_{gen}([k_S], C, S, cp)$	;*Ticket generation*
	$new([k_S], key[], \langle AS[], C, S, Q, [n], cp, P' \rangle)$	;*Nonce* $[k_S]$ *generation*
KRB_AS_REP($K \rightarrow U$)	C,	;*Client name*
	$\langle S, \{[k_S], C, P'\}_{sk(S)} \rangle$	;*Initial Ticket*
	$\{[k_S], [n], P', S\}_{ck(C)}$	;*Session key info*
Phase 2 : Get TGT or ST using TGT (*User* $\leftrightarrow$ *Ticket Granting Server*)		
KRB_TGS_REQ($U \rightarrow G$)	$\langle TGS[], \{[k_S], C, P\}_{sk(TGS[])} \rangle$,	;*Forwarded TGT*
	$\{C, cq, [k]^*\}_{[k_S]}$,	;*Authenticator*
	S, Q, $[n]$	;*Request body*
TGS	$accept([k_S], C, TGS[], cp)$	;*TGT acceptance*
	$init_{gen}([k'_S], C, S, cp)$	;*Ticket generation*
	$new([k'_S], key[], \langle TGS[], [k_S], C, P, cq, [k], S, Q, [n], cp, P' \rangle)$	;*Nonce* $[k'_S]$ *generation*
KRB_TGS_REP($G \rightarrow U$)	C,	;*Client name*
	$\langle S, \{[k'_S], C, P'\}_{sk(S)} \rangle$,	;*New Ticket*
	if $[k]$ presents then $\{[k'_S], [n], P', S\}_{[k]}$	;*Session key info*
	else $\{[k'_S], [n], P', S\}_{[k_S]}$	;*Session key info*
Additional Phase 2 : Renew and Validate TGT or ST (*User* $\leftrightarrow$ *Ticket Granting Server*)		
KRB_TGS_REQ($U \rightarrow G$)	$\langle S, \{[k_S], C, P\}_{sk(S)} \rangle$,	;*Forwarded ticket*
	$\{C, cq, [k]^*\}_{[k_S]}$	;*Authenticator*
	S, Q, $[n]$	;*Request body*
TGS	$accept([k_S], C, S, cp)$	;*Ticket acceptance*
	$init_{gen}([k'_S], C, S, cp)$	;*Ticket generation*
	$new([k'_S], key[], \langle S, [k_S], C, P, cq, [k], Q, [n], cp, P' \rangle)$	;*Nonce* $[k'_S]$ *generation*
KRB_TGS_REP($G \rightarrow U$)	C,	;*Client name*
	$\langle S, \{[k'_S], C, P'\}_{sk(S)} \rangle$,	;*New ticket*
	if $[k]$ presents then $\{[k'_S], [n], P', S\}_{[k]}$	;*Session key info*
	else $\{[k'_S], [n], P', S\}_{[k_S]}$	;*Session key info*
Phase 3 : Get Service using ST (*User* $\leftrightarrow$ *Application Server*)		
KRB_AP_REQ($U \rightarrow P$)	$\langle AP[], \{[k_S], C, P\}_{sk(AP[])} \rangle$,	;*Forwarded ST*
	$\{C, cq, [k]^*\}_{[k_S]}$	;*Authenticator*
AP	$accept([k_S], C, AP[], cp)$	;*ST acceptance*

where $Q = \langle fq^*, tq, rq^* \rangle$, $P, P' = \langle ap, sp^*, ep, rp^* \rangle$ and cp is the time when the server replies.

Table 3. Kerberos Message Exchanges

Name (Flag)	Explanation
RN (RENEWABLE)	If renewable tickets are allowed
RO (RENEWABLE_OK)	When RN is set and the requested ticket does not satisfy policy on $\S l$, if a renewable ticket can be issued instead.
PO (MAY_POSTDATE)	If the ticket can be issued for future usage
$\S l$	The maximum life-time of tickets
$\S r$	The maximum renewable life-time of tickets

Table 4. Quantifiable Parameters in Kerberos

Phase 1, 2 and 3				
Boolean	Input Q	Input P	Output P'	Constraints
RN*, RO*, PO*	fq^*, tq	ap, ep, rp^*	ap', ep'	$cp \leq ep \wedge ep' \leq tq \wedge ep' \leq ep$ if P exists then $ap' = ap$ else $ap' = cp$ $\wedge\ fq \leq cp \leq ep' \leq cp + \S l$
RN*, RO*, PO	fq, tq	ap, ep, rp^*	ap', sp', ep'	$cp \leq ep \wedge ep' \leq tq \wedge ep' \leq ep$ if P exists then $ap' = ap$ else $ap' = cp$ $\wedge\ fq = sp' \wedge cp < fq \leq ep' \leq sp' + \S l$
RN, RO*, PO*	fq^*, tq, rq	ap, ep, rp	ap', ep', rp'	$cp \leq ep \wedge ep' \leq tq \wedge ep' \leq ep$ if P exists then $ap' = ap$ else $ap' = cp$ $\wedge\ fq \leq cp \leq ep' \leq cp + \S l$ $\wedge\ ep' \leq rp' \leq rq \wedge rp' \leq cp + \S r$
RN, RO*, PO	fq, tq, rq	ap, ep, rp	ap', sp', ep', rp'	$cp \leq ep \wedge ep' \leq tq \wedge ep' \leq ep$ if P exists then $ap' = ap$ else $ap' = cp$ $\wedge\ fq = sp' \wedge cp < fq \leq ep' \leq sp' + \S l$ $\wedge\ ep' \leq rp' \leq rq \wedge rp' \leq sp' + \S r$
RN, RO, PO*	fq^*, tq	ap, ep, rp	ap', ep', rp'	$cp \leq ep \wedge ep' \leq ep$ if P exists then $ap' = ap$ else $ap' = cp$ $\wedge\ fq \leq cp \leq ep' \leq cp + \S l < tq$ $\wedge\ ep' \leq rp' \leq tq \wedge rp' \leq cp + \S r$
RN, RO, PO	fq, tq	ap, ep, rp	ap', sp', ep', rp'	$cp \leq ep \wedge ep' \leq ep \wedge fq = sp'$ if P exists then $ap' = ap$ else $ap' = cp$ $\wedge\ cp < fq \leq ep' \leq sp' + \S l < tq$ $\wedge\ ep' \leq rp' \leq tq \wedge rp' \leq sp' + \S r$
Additional Phase 2				
Boolean	Input Q	Input P	Output P'	Constraints
RN*, RO*, PO	fq^*, tq	ap, sp, ep	ap', ep'	$sp \leq cp \leq ep \wedge ep' \leq tq \wedge ap' = ap$ $\wedge\ fq \leq cp \leq ep' \leq cp + \S l \wedge ep' \leq ep$
RN, RO*, PO	fq^*, tq	ap, sp, ep, rp	ap', ep', rp'	$sp \leq cp \leq ep \wedge ep' \leq tq \wedge ap' = ap$ $\wedge\ fq \leq cp \leq ep' \leq cp + \S l$ $\wedge\ ep' \leq ep \wedge rp' = rp$
RN, RO*, PO*	fq^*, tq	ap, ep, rp	ap', ep', rp'	$cp \leq ep \wedge ep' \leq tq \wedge ap' = ap$ $\wedge\ fq \leq cp \leq ep' \leq cp + \S l \wedge rp' = rp$

(1) In Phase 1, the Input P is absent. Similarly, both of the Q and the P' are absent in Phase 3.
(2) The timestamps superscripted by '*' are optional.

Table 5. Timing Constraints Checked by Servers

– If $B \cap L \neq \emptyset$, $\Pi(T, L)$ becomes a derivation tree of $\pi(R, L)$ based on $\alpha(\mathbb{R}_{init}, L)$, and $\Pi(T', L)$ becomes a derivation tree of $\pi(R, L)$ based on $\alpha(\beta(\mathbb{R}_{init}), L)$.
– If $B \cap L = \emptyset$, $\pi(R, L)$ becomes invalid, so both of $\Pi(T, L)$ and $\Pi(T', L)$ do not exist.

Hence, let $R' = \pi(R, L)$, R' is derivable from $\alpha(\mathbb{R}_{init}, L)$ if and only if R' is derivable from $\alpha(\beta(\mathbb{R}_{init}), L)$. The theorem is then proved.

Theorem 2. Partial Correctness. Let Q be the query and $\mathcal{R}_{init}$ be the initial rule set. There exists R derivable from $\alpha(\mathcal{R}_{init}, L)$ such that $Q \nvdash R$ if and only if there exists $R' \in \alpha(\beta(\mathbb{R}_{init}), L)$ such that $Q \nvdash R'$.

Proof. **Partial Soundness.** Given $R' \in \alpha(\beta(\mathbb{R}_{init}), L)$, according to Theorem 1, R' is derivable from $\alpha(\mathbb{R}_{init}, L)$. Hence, there exists R' derivable from $\alpha(\mathcal{R}_{init}, L)$ such that $Q \nvdash R'$. **Partial Completeness.** Given a rule R derivable from $\alpha(\mathcal{R}_{init}, L)$ such that $Q \nvdash R$, according to Theorem 1, there exists a rule $R_0 = H_0 \dashv\!\lfloor\ B_0\ \rfloor\!\mapsto e_0$ derivable from $\alpha(\beta(\mathbb{R}_{init}), L)$ such that $Q \nvdash R_0$. So there exists a derivation tree of R_0 whose nodes are labeled by rules in $\alpha(\beta(\mathbb{R}_{init}), L)$. We prove that the rule $R_t = H_t \dashv\!\lfloor\ B_t\ \rfloor\!\mapsto e_t$ labeled on the tree's root is also a contradiction as follows.

– If Q is a secrecy query, R_t has a leak event as conclusion because $Q \nvdash R_0$. Additionally, since $R_t \in \alpha(\beta(\mathbb{R}_{init}), L)$, $\forall e'_t \in H_t, e'_t \in \mathbb{V}$. Thus, $Q \nvdash R_t$.
– If $Q = e_q \leftarrow\!\lfloor\ B_q\ \rfloor\!\vdash H_q$ is a timed authentication query, e_t is an accept event. So, R_t should satisfy either $Q \vdash R_t$ or $Q \nvdash R_t$. If $Q \vdash R_t$, as all of the arguments in e_q are variables, there exists a most general unifier σ of e_t and e_q satisfying $\sigma e_q = e_t$, and $\exists \sigma', (\sigma' \sigma H_q \subseteq \sigma H_t) \wedge (\sigma B_t \subseteq \sigma' \sigma B_q)$ Meanwhile, incoming edges of the tree root cannot be init events and new events, so these events should also persist in R_0. Hence, $Q \vdash R_0$. This violates our precondition that $Q \nvdash R_0$. We then have $Q \nvdash R_t$.

Appendix C: Extensions to the Dolev-Yao model

In addition to the attacker capabilities in the Dolev-Yao model, the attacker can compromise cryptographic primitives and legitimate protocol participants. For instance, the attacker may be able to conduct the brute-force attack on the encrypted messages, when a weak encryption function is used. Given the name of the encryption function as *Crypto* and the least time of cracking *Crypto* is $\S d$, the attacking behavior can be modeled by the following rule.

$$know(Crypto(m, k), t_1) \dashv\!\lfloor\ t - t_1 > \S d\ \rfloor\!\mapsto know(m, t)$$

Additionally, some ciphers like RC4 which is used by WEP, key compromise on a busy network can be conducted after a short time. Given an application scenario

where such attack is possible and the attacking time has a lower bound $\S d$, we can model it as follows.

$$know(RC4(m,k),t_1)\rangle \dashv\!\lfloor\, t - t_1 > \S d \,\rfloor\!\mapsto know(k,t)$$