TAuth: Verifying 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 relevant to the security of systems, like web applications, cyber-physical systems, etc. Verifying timed security protocols is however challenging as both arbitrary attacking behaviors and quantitative timing may lead to undecidability. In this work, we develop a service framework to support intuitive modeling of the timed protocol, as well as automatic verification with an unbounded number of sessions. The partial soundness and completeness of our verification algorithms are formally defined and proved. We implement our method into a tool called TAuth and the experiment results show that our approach is efficient and effective in both finding security flaws and giving proofs.

1 Introduction

Timed security protocols are used extensively nowadays. Many security applications [29, 9, 4] use time to guarantee the freshness of messages received over the network. In these applications, messages are associated with timing constraints so that they can only be accepted in a predefined time window. As a result, relaying and replaying messages are allowed only in a timely fashion. It is known that security protocols and their manual proofs are error-prone, which has been evidenced by multiple flaws found in existing proved protocols [30, 27, 17]. It is therefore important to have automatic tools to formally verify these protocols.

However, existing methods and tools for security protocol verification often abstract timestamps away by replacing them with nonces. The main reason is that most of the decidability results are given for untimed protocols [24, 28]. Thus, the state-of-the-art security protocol verifiers, e.g., ProVerif [6], Athena [31], Scyther [13] and Tamarin [25], are not designed to specify and verify time sensitive cryptographic protocols. Abstracting time away may lead to several problems. First, since the timestamps are abstracted as nonces, the message freshness checking in the protocol cannot be correctly specified. As a consequence, attacks found in the verification may be false alarms because they could be impractical when the timestamps are checked. Second, omitting the timestamp checking could also result in missing attacks. For instance, the timed authentication property ensures the satisfaction of the timing constraints in addition to the establishment of the event correspondence. Without considering the timing constraints, even though the agreement is verified under the untimed configuration correctly, the protocol may still be vulnerable to timing attacks. Third, with light-weight encryption, which are often employed in cyber-physical systems, it might be possible to decrypt secret messages in a brute-force manner given sufficient time. In applications where long network latency is expected, it is therefore essential to consider timing constraints explicitly and check the feasibility of attacks.

Contributions. In this work, we provide a fully automatic approach to verify timed security protocols with an unbounded number of sessions. Our contributions are fourfold. (1) In order to precisely specify the capabilities of the adversary, we propose a service framework in which the adversary's capabilities are modeled as various services according to the protocol specification and cryptographic primitives. Thus, when the protocol is vulnerable, there should exist an attack trace consisting of the services in a certain sequence. (2) An automatic algorithm is developed in this work to verify the timed authentication properties with an unbounded number of sessions. Since security protocol verification is undecidable in general [11], we cannot guarantee the termination of our algorithm. We thus define partial soundness and completeness in Section 2.3 and prove that our algorithm is partially sound and complete in Section 3. (3) Having time in security protocol verification adds another dimension of complexity. Thus we propose the finite symbolic representation for the timing constraints with approximation. We prove that the protocol is guaranteed to be secure when it is full verified by our algorithm. Additionally, when the protocol specification is in a specific form, we also prove that our algorithm does not introduce false alarms. (4) A verifier named TAuth is developed based on our method. We evaluate TAuth using several timed and untimed security protocols [8, 10, 26, 19, 7, 29]. The experiment results show that our approach is efficient and effective in both finding security flaws and giving proofs.

Related works. Evans et al. [16] introduced a semi-automated way to analyze timed security protocols. They modeled the protocols with CSP and checked them with PVS. In [23], Lowe proposed finite state model checking to verify bounded timed authentication. In order to avoid the state space explosion problem, protocol instances and time window are bounded in the verification. Jakubowska et al. [18] and Corin et al. [12] used Timed Automata to specify the protocols and used Uppaal to give bounded verifications. Our method is different from theirs as our verification algorithm is fully automatic and the verification result is given for an unbounded number of sessions.

The work 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 to [14], the protocol specification is modified by explicitly encoding an additional timestamp, which represents the protocol initialization time, into some messages. Thus the attack could be found by comparing that timestamp with the original timestamps in the messages. However, it is not clearly illustrated in their paper how their approach can be applied to timed security protocol verification in general. On the other hand, our approach could be directly applied to crypto protocols without any manual modification to the protocol specification.

We adopt the horn logic which is similar to the one used in ProVerif [6], a very efficient security protocol verifier designed for untimed cryptographic protocol, and extend it with timestamps and timing constraints. However, the extension for time is nontrivial. In ProVerif, the fresh nonces are merged under the same execution trace, which is one of major reasons for its efficiency. When time is involved in the protocol, the generation time of the nonces in the protocol becomes important for the verification. Thus merging the session nonces under the same execution trace often introduces false

Type	Expression	
Timestamp(t)	t	
Message(m)	$g(m_1, m_2, ..., m_n)$	(function)
	$a[]$	(name)
	$[n]$	(nonce)
	v	(variable)
	t	(timestamp)
Fact(f)	$\langle m, t \rangle$	(timed communication)
	$e(m_1, m_2, ..., m_n)$	(event)
Constraint($\mathcal{B}$)	$\mathcal{C}(t_1, t_2, ..., t_n)$	
Service(S)	$f_1, f_2, ..., f_n \dashv\!\![\, \mathcal{B} \,]\!\!\mapsto f$	
Query($\mathcal{Q}$)	$accept(\ldots) \leftarrow\!\![\, \mathcal{B} \,]\!\!\vdash init_1(\ldots), \ldots, init_n(\ldots)$	

Table 1. Service Syntax Hierarchy

alarms into the verification results. In order to differentiate the nonces generated in the sessions, we encode the session nonces into the events engaged in the protocol and use the events to distinguish them. Additionally, our approach takes care of the infinite expansion of timing constraints, which is discussed in Section 3.1.

2 Protocol Specification Framework

We introduce the proposed protocol specification framework in this section. In the framework, the security protocols and the cryptographic primitives are modeled as various services accessible to the *Adversary* for conducting attacks. Generally, these services receive inputs from the adversary and send the results back to the adversary as output over the network. Timestamps are tagged to the messages to denote when they are known to the adversary. We assume the adversary model presented in this framework is an active attacker who can intercept all communications, compute new messages and send any messages he obtained. For instance, he can use all the publicly available functions including encryptions, decryptions, concatenations, etc. He can also ask legal protocol participants to take part in the protocol when he needs. Thanks to the introduction of time, key expiration and message compromise can also be specified by adding additional services.

2.1 Service Syntax

In our framework, services are represented by a set of horn logic rules guarded by timing constraints. We adopt the syntax shown in Table 1 to define the services. *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 values in sessions; *variables* are memory spaces for holding *messages*; and *timestamps* are values extracted from the global clock during the protocol execution. A *fact* can be a *message* tagged by a *timestamp* denoted as $\langle m, t \rangle$, which means that the message m is known to the adversary at time t. Otherwise, it is an *event* in the form of $e(m_1, \ldots, m_n)$ where e is the event name and $m_1, \ldots, m_n$ are the event arguments. The events are used for specifying authentication properties and distinguishing different sessions. $\mathcal{B}$ is a set of closed timing constraints assigned on the *timestamp* pairs. Each constraint is in the form of $t - t' \sim d$ where t and t' are *timestamps*, d is an integer constant (∞ is omitted), and $\sim$ denotes either $<$ or $\leq$. We denote the maximum value of d in a timing constraint set $\mathcal{B}$ as $max(\mathcal{B})$. For simplicity, when a timing constraint $t - t' \sim d \in \mathcal{B}$, we write $d(\mathcal{B}, t, t')$ to denote the integer constant d, and $c(\mathcal{B}, t, t')$ to denote the comparator $\sim$[4]. A *service* $f_1, f_2, ..., f_n \dashv\![\, \mathcal{B} \,]\!\mapsto f$ means that if the *facts* $f_1, f_2, ..., f_n$ and the constraints $\mathcal{B}$ are satisfied, the adversary can invoke this service and obtain f as the result.

2.2 Service Modeling

In the following, we show how to model the timed authentication protocols in our framework. We illustrate the service modeling using a simple example called the Wide Mouthed Frog (WMF) protocol [8] as described below.

$$\begin{aligned} A \rightarrow S :& A, \{t_A, B, k\}_{k_A} \\ S \rightarrow B :& \{t_S, A, k\}_{k_B} \end{aligned}$$

In the protocol, A and B are two users *Alice* and *Bob*, and S is a trust server who shares different secret keys with different users. The goal of this protocol is to share a fresh key k from *Alice* to *Bob*. k_A is the secret key shared between server and *Alice*, and k_B is the corresponding secret key for *Bob*. k is a fresh session key generated by *Alice*, which should be different in different sessions. t_A is a timestamp generated by *Alice*. Similarly, t_S is a timestamp generated by the server. In the protocol, we assume that the clock drift for every participants is negligible, so that the message freshness checking is valid during the execution.

When the server receives the request from *Alice*, it checks its freshness by comparing the t_A with the current clock reading t_S. If t_A and t_S satisfy the pre-defined constraint C_1, the server then sends the second message to *Bob*. Upon receiving the message from the server, *Bob* decrypts it and compares t_S with his clock reading t_B. If the timestamp checking C_2 is passed and the message is properly formed, *Bob* then believes that k is a fresh key shared with *Alice*. In fact, there exists an attack [3] to the protocol which is resulted from the symmetric structure of the exchanged messages.

$$\begin{array}{rl} A \rightarrow S : & A, \{t_A, B, k\}_{k_A} \\ S \rightarrow I(B) : & \{t_S, A, k\}_{k_B} \\ I(B) \rightarrow S : & B, \{t_S, A, k\}_{k_B} \\ S \rightarrow I(A) : & \{t_{S'}, B, k\}_{k_A} \\ I(A) \rightarrow S : & A, \{t_{S'}, B, k\}_{k_A} \\ S \rightarrow B : & \{t_{S''}, A, k\}_{k_B} \end{array}$$

[4] If a timing constraint is not specified exactly in this form, it should be possible to change the constraint into this form. For instance, $t - t' > 3$ can be changed into $t' - t < -3$.

In the attack trace, the adversary I personates *Bob*, hijacks the second message and sends it back to the server within the timing constraint C_1. Then, the server would treat it as a valid request from *Bob* and update the t_S to its current clock reading. By doing this repeatedly, the timestamp in the request can be extended to an arbitrary large value. As a result, when *Bob* receives a message that passes the timestamp checking, the request from *Alice* may not be timely any more. Hereafter, we assume that the server and *Bob* check the freshness of the received messages with following timing constraints: $C_1 = t_S - t_A \leq 2$ and $C_2 = t_B - t_S \leq 2$. Notice that in general, the constraints should be set according to the protocol specification, network latency, etc.

Crypto Services. Cryptographic primitives are usually specified as services without network latency. Generally, we have two types of crypto services, which are constructors and destructors. Constructors are used to generate new messages such as concatenation and encryption, whereas destructors are used to extract messages from the constructed messages. For instance, the constructor and the destructor for symmetric encryption can be modeled as follows.

$$\langle m, t_1\rangle, \langle k, t_2\rangle -\!\![\, t_1 \leq t \wedge t_2 \leq t \,]\!\!\rightarrow \langle enc_s(m,k), t\rangle \tag{1}$$

$$\langle enc_s(m,k), t_1\rangle, \langle k, t_2\rangle -\!\![\, t_1 \leq t \wedge t_2 \leq t \,]\!\!\rightarrow \langle m, t\rangle \tag{2}$$

The service (1) means that if the adversary has a message m and a key k, this service can generate the symmetric encryption for m by k, and the timing t of receiving the encryption should be later than the timing t_1 and t_2 when m and k are known to the adversary. The symmetric decryption service is similarly defined in service 2.

For some cryptographic primitives, additional constraints can be added for special purposes. For instance, RSA encryption may consume non-negligible time to compute. If the encryption time has a lower bound d, we could use the following constructor to model the additional requirement on time.

$$\langle m, t_1\rangle, \langle pk, t_2\rangle -\!\![\, t - t_1 > d, t - t_2 > d \,]\!\!\rightarrow \langle RSA(m, pk), t\rangle$$

Protocol Services. Protocol services are used to specify the execution of the protocol. These services are directly derived from the protocol specification. Specifically, for the WMF protocol, the server S answers queries from all its users. After receiving a request from a user I, S extracts the message content and checks the timestamp. If the timestamp is generated within 2 time units, S sends out the encryption of an updated timestamp t_S, the initiator's name and the session key k under the responder's shared key. The service provided by the server can be specified with

$$\begin{aligned}\langle enc_s((t_I, R, k), key(I)), t\rangle, \langle I, t'\rangle -\!\![\, 0 \leq t_S - t_I \leq 2 \wedge t \leq t_S \wedge t' \leq t_S \,]\!\!\rightarrow \\ \langle enc_s((t_S, I, k), key(R)), t_S\rangle\end{aligned} \tag{3}$$

in which $key(U)$ represents the secret key shared between the server and the user U. Since the keys are only shared with the user and the server, We do not treat the key constructor as a public service. Besides, the names of the two participants should be known to the adversary, so we have services for publishing their names.

$$-\!\![\quad]\!\!\rightarrow \langle A[], t\rangle \tag{4}$$

$$-\!\![\quad]\!\!\rightarrow \langle B[], t\rangle \tag{5}$$

Event Services. In order to ensure the authenticity between participants, we introduce two special events *init* and *accept*. The *init* event is explicitly engaged by the adversary when he wants to start a new protocol session, while the *accept* event is engaged by the protocol when the timed authentication is established successfully. According to [22], the timed authentication is correct if and only if every *accept* event is emitted with its corresponding *init* event engaged before, and the timing constraints should always be satisfied. For the WMF protocol, the adversary engages an event *init* when he wants *Alice* to start a session with R.

$$\mathit{init}(A[], R, [k], t_A) \dashv\![\quad]\!\!\mapsto \langle \mathit{enc}_s((t_A, R, [k]), \mathit{key}(A[])), t_A\rangle \tag{6}$$

When the user *Bob* gets the message from the server, he decrypts it with his shared key $\mathit{key}(B[])$ and checks its freshness. If the timestamp checking is passed and the initiator is I, he then believes that he has established a timely authenticated connection under session key k with I and engages an *accept* event as follows.

$$\langle \mathit{enc}_s((t_S, I, k), \mathit{key}(B[])), t\rangle \dashv\![\, t_B - t_S \leq 2 \,]\!\!\mapsto \mathit{accept}(I, B[], k, t_B) \tag{7}$$

Additional Services. Introducing time allows to model systems which are not possible previously. For instance, some applications require that the passwords are used only if they are unexpired. One possible scenario is that the token $\mathit{token}(s, pw, t_k)$ can only be opened within the lifetime $[t_k, t_k + d]$ of the password pw.

$$\langle \mathit{token}(s, pw, t_k), t_1\rangle, \langle pw, t_2\rangle \dashv\![\, t_k \leq \left\{\begin{matrix} t_1 \\ t_2 \end{matrix}\right\} \leq \left\{\begin{matrix} t_k + d \\ t \end{matrix}\right\}]\!\!\mapsto \langle s, t\rangle$$

If the adversary can obtain both of the token and the password within $[t_k, t_k + d]$, the secret s can be extracted from the token. Another possible service that could be accessible to the adversary is the brute force attack on the encrypted messages, which allows the adversary to extract the encrypted data without knowing the key. Suppose the least time of cracking the crypto is d, the attacking behavior can be modeled with

$$\langle \mathit{Crypto}(m, k), t\rangle \dashv\![\, t' - t > d \,]\!\!\mapsto \langle m, t'\rangle.$$

For 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 d, we can model it as follows.

$$\langle \mathit{RC4}(m, k), t\rangle \dashv\![\, t' - t > d \,]\!\!\mapsto \langle k, t'\rangle$$

Remarks. Even though the services specified in our framework can directly extract the message from the encryption without the key and so on, a given protocol can still guarantee correctness as long as proper timing checking is in place, e.g., authentication should be established before the adversary has the time to finish the brute-force attack.

2.3 Security Properties

In this work, we focus on verifying that the authentication between the two participants is timely, which means every $accept$ event is preceded by a corresponding $init$ event satisfying the timing constraints. Thus we formalize the *timed authentication* property by extending the definition in [22] as follows.

Definition 1. ***Timed Authentication.*** *In a timed security protocol, timed authentication holds for an* accept *event f with a set of* init *events H agreed on arguments encoded in the events and the timing constraints $\mathcal{B}$, if and only if for every occurrence of f, all of the corresponding* init *events in H should be engaged before, and their timestamps should always satisfy the timing constraints $\mathcal{B}$. We denote the* timed authentication *query as $f \leftarrow\!\dashv\![\; \mathcal{B} \;]\!\vdash H$. In order to ensure general timed authentication, the arguments encoded in events should only be different variables and timestamps.*

We remark that the *timed authentication* defined above is the non-injective agreement. Since injective agreement is usually implemented by duplication checking, which is unrelated to time, we do not discuss *injective timed authentication* [22] in this work. Because the legitimate run of WMF protocols requires that the authentication should be established within 4 time units, its query is modeled as follows.

$$accept(I, R, k, t) \leftarrow\!\dashv\![\; t - t' \leq 4 \;]\!\vdash init(I, R, k, t') \tag{8}$$

In Section 3, we present a verification algorithm to check the authentication. Since the verification for security protocol is generally undecidable [11], our algorithm cannot guarantee termination. Hence, we claim our attack searching algorithm as partial sound and partial complete under the condition of termination (partial correctness).

3 Verification Algorithm

Given the specification formalized in Section 2, our verification algorithm is divided into two phases. The attack searching service basis is constructed in the first phase so that attacks can be found in a straight forward method in the second phase. Specifically, every service consists of several inputs, one output and some timing constraints. When a service's input can be provided by another service's output, we could compose these two services together to form a composite service. In the first phase, our algorithm composes the services repeatedly until a fixed-point is reached. When such a fixed-point exists, we call it the *guided service basis*. However, the above process may not terminate because of two reasons. The first reason is the infinite knowledge deduction. For example, given two services $m \dashv\![]\!\mapsto h(m)$ and $h(m) \dashv\![]\!\mapsto h(h(m))$, we can compose them to obtain a new service $m \dashv\![]\!\mapsto h(h(m))$, which could be composed to the second service again. In this way, infinitely many composite services can be generated. The second reason is the infinite expansion of timing constraints. For instance, assume we have $S_0 = \langle enc(t', k), t_1 \rangle \dashv\![\; t''-t' \leq 2 \wedge t_1 \leq t'' \;]\!\mapsto \langle enc(t'', k), t'' \rangle$ and $S_1 = init(t, [k]) \dashv\![\; t' - t \leq 2 \wedge t \leq t' \;]\!\mapsto \langle enc(t', [k]), t' \rangle$ in the service basis. When we compose S_1 to S_0, their composition $S_2 = init(t, [k]) \dashv\![\; t'' - t \leq 4 \wedge t \leq t'' \;]\!\mapsto \langle enc(t'', [k]), t'' \rangle$ has a larger range than S_1. Besides, we could compose S_2 to S_0 again to obtain an even larger range, so the service composition never ends. Since verification for untimed security protocol is undecidable, we, same as state-of-the-art tools like ProVerif, cannot handle the first scenario. We thus focus on solving the second scenario by approximating the timing constraints into a finite set. The fixed-point is then called the *approximated service basis*. When the over-approximation is applied, false alarms may be introduced into the verification result so that, generally, only partial completeness is preserved by our attack searching algorithm. Finally, we present our attack searching algorithm in the end of this section.

3.1 Service Basis Construction

In the first phase, our goal is to construct a set of services that allows us to find security attacks in the second phase. In order to construct such a service basis, new services are generated by composing existing services. In this way, the new composite services can also be treated as services directly accessible to the adversary and the algorithm continues until the fixed-point is reached, i.e., no new service can be generated. We use the most general unifier to unify the input and the output.

Definition 2. ***Most General Unifier.*** *If σ is a substitution for both messages m_1 and m_2 so that $\sigma m_1 = \sigma m_2$, we say m_1 and m_2 are unifiable and σ is an unifier for m_1 and m_2. If m_1 and m_2 are unifiable, the most general unifier for m_1 and m_2 is an unifier σ such that for all unifiers σ' of m_1 and m_2 there exists a substitution σ'' such that $\sigma' = \sigma''\sigma$.*

Since the adversary in our framework has the capability to generate new names and new timestamps, when a service input is a variable or a timestamp that is unrelated to other facts in a service, the adversary should be able to generate a random fact and use it to fulfill that input. In this way, that input can be removed in the composite service. Hence, we define service composition as follows. For simplicity, we define a *singleton* as a fact of the form $\langle x, t\rangle$ where x is a variable or a timestamp.

Definition 3. ***Service Composition.*** *Let $S = H \mathrel{-\!\![}\ \mathcal{B}\ \mathrel{]\!\!\rightarrow} f$ and $S' = H' \mathrel{-\!\![}\ \mathcal{B}'\ \mathrel{]\!\!\rightarrow} f'$ be two services. Assume there exists $f_0 \in H'$ such that f and f_0 are unifiable, their most general unifier is σ and $\sigma\mathcal{B} \cap \sigma\mathcal{B}' \neq \phi$. The service composition of S with S' on a fact f_0 is defined as*

$$S \circ_{f_0} S' = \mathit{clear}(\sigma(H \cup (H' - \{f_0\}))) \mathrel{-\!\![}\ \mathit{sim}(\sigma\mathcal{B} \cap \sigma\mathcal{B}')\ \mathrel{]\!\!\rightarrow} \sigma f'$$

where the function clear merges duplicated facts from the inputs and removes any singleton $\langle x, t\rangle$ where x does not appear in other facts of the rule, and the function sim removes timestamps that are no longer used in the composite service.

When new composite services are added into the service basis, redundancies should be eliminated from the service basis. As the timing constraints can be viewed as a set of clock valuations which satisfy the constraints, they thus can be naturally applied with set operations, e.g., $\mathcal{B} \subseteq \mathcal{B}'$, $\mathcal{B} \cap \mathcal{B}'$, etc.

Definition 4. ***Service Implication.*** *Let $S = H \mathrel{-\!\![}\ \mathcal{B}\ \mathrel{]\!\!\rightarrow} f$ and $S' = H' \mathrel{-\!\![}\ \mathcal{B}'\ \mathrel{]\!\!\rightarrow} f'$ be two services. S implies S' denoted as $S \Rightarrow S'$ if and only if $\exists\sigma, \sigma f = f' \wedge \sigma H \subseteq H' \wedge \mathcal{B}' \subseteq \sigma\mathcal{B}$.*

When services are composed in an unlimited way, infinitely many composite services could be generated. For instance, composing the symmetric encryption service (1) to itself on the fact $\langle m, k\rangle$ leads to a new service encrypting the message twice, that is $\langle m, t\rangle, \langle k_1, t_1\rangle, \langle k_2, t_2\rangle \dashv\!\![\ldots]\!\!\mapsto \langle enc_s(enc_s(m, k_1), k_2), t'\rangle$, which can be composed to the encryption service again. In order to avoid these service compositions, we adopt a similar strategy proposed in [6] such that the unified fact in the service composition should not be singletons. Moreover, the events in our system cannot be unified[5], thus we define $\mathcal{V}$ as a set of facts that should not be unified, consisting of all events and singletons.

We denote $\beta(\alpha, \mathcal{R}_{init})$ as the fixed-point, where $\mathcal{R}_{init}$ is the initial service set and α is a service approximation function adopted during the construction. In order to compute $\beta(\alpha, \mathcal{R}_{init})$, we first define $\mathcal{R}_v$ based on the following rules, where $inputs(S)$ represents the inputs of service S.

1. $\forall S \in \mathcal{R}_{init}, \exists S' \in \mathcal{R}_v, S' \Rightarrow S$;
2. $\forall S, S' \in \mathcal{R}_v, S \not\Rightarrow S'$;
3. $\forall S, S' \in \mathcal{R}_v$, if $\forall f_{in} \in inputs(S), f_{in} \in \mathcal{V}$ and $\exists f \notin \mathcal{V}, S \circ_f S'$ is defined, $\exists S'' \in \mathcal{R}_v, S'' \Rightarrow \alpha(S \circ_f S')$.

The first rule means that every initial service is implied by a service in $\mathcal{R}_v$. The second rule means that no duplicated service exists in $\mathcal{R}_v$. The third rule means that for any two services in $\mathcal{R}_v$, if the first service's inputs are in $\mathcal{V}$ and their composition exists, their approximated composition is also implied by a service in $\mathcal{R}_v$. These three rules means $\mathcal{R}_v$ is the minimal closure of the initial service set $\mathcal{R}_{init}$. Based on $\mathcal{R}_v$, we have

$$\beta(\alpha, \mathcal{R}_{init}) = \{S \mid S \in \mathcal{R}_v \wedge \forall f_{in} \in inputs(S) : f_{in} \in \mathcal{V}\}.$$

In the latter part of this section, α will be instantiated with no-approximation and over-approximation. (The detailed algorithm is available in Appendix C.)

For any service, it is derivable from a service basis $\mathcal{R}$ if and only if there is a derivation tree that represents how the service is composed.

Definition 5. ***Derivation Tree.*** *Let $\mathcal{R}$ be a set of closed services and S be a closed service, where a closed service is a service with its output initiated by its inputs. Let S be a service in the form of $f_1, \ldots, f_n \dashv\!\![\mathcal{B}]\!\!\mapsto f$. S can be derived from $\mathcal{R}$ if and only if there exists a finite derivation tree defined as*

1. *edges in the tree are labeled by facts;*
2. *nodes are labeled by the services in $\mathcal{R}$;*
3. *if a node labeled by S has incoming edges of $f^s_1, \ldots, f^s_n$, an outgoing edge of f^s, and the timestamps among these facts satisfy the timing constraints $\mathcal{B}^s$, then $S \Rightarrow f^s_1, \ldots, f^s_n \dashv\!\![\mathcal{B}^s]\!\!\mapsto f^s$;*
4. *the outgoing edge of the root is the fact f;*
5. *the incoming edges of the leaves are $f_1, \ldots, f_n$.*

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

[5] *init* events only appear in the inputs and *accept* events only appear in the output.

Guided Service Basis. When no approximation is used in the service basis construction, the fixed-point is called *guided service basis* denoted as $\mathcal{R}_{guided} = \beta(\alpha_{guided}, \mathcal{R}_{init})$ where, for any service S, $\alpha_{guided}(S) = S$. In such a case, we prove that a service can be derived from the guided service basis whenever it can also be derived from the initial service set, and vice versa.

Theorem 1. *For any service S in the form of $H \dashv\!\![\, \mathcal{B} \,]\!\!\mapsto f$ where $\forall f_{in} \in H : f_{in} \in \mathcal{V}$, S is derivable from $\mathcal{R}_{init}$ if and only if S is derivable from $\mathcal{R}_{guided}$.*

Proof Sketch. **Only if.** Given a service, if it is derivable from the initial service set, its derivation tree should exist. We thus compose the directly connected nodes in its derivation tree and show that the new composite node is implied by a service in $\mathcal{R}_v$. When no directly connected nodes can be composed, we then prove that the rest of the nodes are labeled by services in $\mathcal{R}_{guided}$, which implies that this service is also derivable from $\mathcal{R}_{guided}$. **If.** On the other hand, $\mathcal{R}_{guided}$ does not introduce extra services except for services derivable from $\mathcal{R}_{init}$, so the theorem is proved. The detailed proof is available in Appendix A.

Approximated Service Basis. New timestamps are often introduced in the service composition. When no longer used timestamps are removed from the composite service, the timing constraints can be deemed as extended for unification. On the other hand, given two services with the same inputs and output but they have different timing constraints, they may be indifferent if all of the different constraints have exceeded a ceiling. For instance, if the password has a fixed lifetime, its usefulness for the adversary remains the same when the password has already expired. Since these services can be deemed as the same, we remove their exceeded timing constraints to generalize their expressiveness. In this work, heuristically, we assume that every service is very likely to be used by the adversary for at least once in the attack trace and the timing constraints in the query also play important role in the reachability checking, so we set the ceiling as $1 + \sum max(\mathcal{B})$ in which $\mathcal{B}$ comes from the initial service set and the query. For instance, in the WMF protocol, the $max(\mathcal{B})$ is 2 for both of the service (3) and (7), 0 for other initial services, and 4 for the query, so we have the ceiling set as 9. We refer to the set of services with the ceiling U as approximated service basis $\mathcal{R}_{approx} = \beta(\alpha^{U}_{approx}, \mathcal{R}_{init})$. The service approximation function α^{U}_{approx} is defined as follows.

Definition 6. ***Service approximation with ceiling** U. Let $S = H \dashv\!\![\, \mathcal{B} \,]\!\!\mapsto f$. We define the service approximation with ceiling U as $\alpha^{U}_{approx}(S) = H \dashv\!\![\, \mathcal{B}' \,]\!\!\mapsto f$. For any two timestamps t, t' in the service S, if $d(\mathcal{B}, t, t') \leq U$, then $d(\mathcal{B}', t, t') = d(\mathcal{B}, t, t')$ and $c(\mathcal{B}', t, t') = c(\mathcal{B}, t, t')$; else if $d(\mathcal{B}, t, t') > U$, then $d(\mathcal{B}', t, t')$ is ∞ and $c(\mathcal{B}', t, t')$ is $<$.*

Since the timing constraints are enlarged after the approximation, false alarms may be introduced into verification result. However, according to the experiment results shown in Section 4, the false alarms could be prevented in the real timed protocol verification when the ceiling is properly configured. On the other hand, whenever a timed protocol is verified as correct under the approximation, it is guaranteed to be attack-free, which is the same as ProVerif.

Theorem 2. *Let U be the ceiling. For any service S in the form of $H \dashv\![\ \mathcal{B}\ ]\!\!\mapsto f$ where $\forall f_{in} \in H : f_{in} \in \mathcal{V}$, if S is derivable from $\mathcal{R}_{init}$, S is also derivable from $\mathcal{R}_{approx}$.*

Proof Sketch. Since the timing constraints are only enlarged in the service basis, according to Theorem 1, it is clear that Theorem 2 also holds. Due to the limitation of the space, we move the detailed proof into Appendix A.

3.2 Query Searching

When the query is violated by a service in the service basis, we call it a contradiction to the query. A service is a contradiction to the query if and only if its output event can be unified to the query's output, while it does not require all the predicate events in the query or it has a larger timing range than the query constraints. Thus, the contradiction is defined as follows.

Definition 7. ***Contradiction.*** *A service $S = H \dashv\![\ \mathcal{B}\ ]\!\!\mapsto f$ is a contradiction to the query $\mathcal{Q} = f' \hookleftarrow\![\ \mathcal{B}'\ ]\!\vdash H'$ if and only if f and f' are unifiable with the most general unifier σ and $\forall \sigma', \sigma'\sigma H' \not\subseteq \sigma H \vee \sigma\mathcal{B} \not\subseteq \sigma'\sigma\mathcal{B}'$.*

If we rewrite the query $\mathcal{Q}$ into a service of $S_q = H' \dashv\![\ \mathcal{B}'\ ]\!\!\mapsto f'$, S is a contradiction to $\mathcal{Q}$ if and only if f' and f are unifiable with the most general unifier σ and we have $\sigma S_q \not\Rightarrow \sigma S$. According to Definition 1, events in the query only contain variables and timestamps that are different. Thus the *accept* event in S_q can be unified with any other *accept* event. The contradiction checking could then be simplified to check whether S outputs an *accept* event and satisfies $S_q \not\Rightarrow S$. Given the service basis $\mathcal{R}$, we thus search the attacks as follows. (The detailed algorithm is available in Appendix C.)

$$\mathcal{R}_f = \{S | S \in \mathcal{R}, \text{the output of } S \text{ is an } \textit{accept} \text{ event} \wedge S_q \not\Rightarrow S\}$$

$\mathcal{R}_f$ consists of the contradiction instances. We prove its partial correctness as follows.

Theorem 3. ***Partial Soundness.*** *Assume $\mathcal{R}$ is $\mathcal{R}_{guided}$. Let $\mathcal{Q}$ be a query of $f' \hookleftarrow\![\ \mathcal{B}'\ ]\!\vdash H'$ and $S_q = H' \dashv\![\ \mathcal{B}'\ ]\!\!\mapsto f'$. There exists S derivable from $\mathcal{R}_{init}$ such that S is a contradiction to $\mathcal{Q}$ if there exists $S' \in \mathcal{R}$ such that the output of S' is an accept event and $S_q \not\Rightarrow S'$.*

Proof Sketch. According to Theorem 1, we have $\forall S' \in \mathcal{R}_{guided}$, S' should be derivable from $\mathcal{R}_{init}$, so any contradiction found in $\mathcal{R}_f$ is valid when $\mathcal{R}$ is $\mathcal{R}_{guided}$.

Theorem 4. ***Partial Completeness.*** *Assume $\mathcal{R}$ is either $\mathcal{R}_{guided}$ or $\mathcal{R}_{approx}$. Let $\mathcal{Q}$ be a query of $f' \hookleftarrow\![\ \mathcal{B}'\ ]\!\vdash H'$ and $S_q = H' \dashv\![\ \mathcal{B}'\ ]\!\!\mapsto f'$. There exists S derivable from $\mathcal{R}_{init}$ such that S is a contradiction to $\mathcal{Q}$ only if there exists $S' \in \mathcal{R}$ such that the output of S' is an accept event and $S_q \not\Rightarrow S'$.*

Proof Sketch. We need to prove that we can find the attack whenever it exists. Since for any service there exists a derivation tree labeled by services in $\mathcal{R}_{guided}$ ($\mathcal{R}_{approx}$ resp.) according to Theorem 1 (Theorem 2 resp.), we prove that if the service S is a contradiction to the query, the service S_r labeled to the root of the tree is also a contradiction to the query. The theorem is thus proved. Due to the limitation of the space, the detailed proofs for the above theorems are available in Appendix A.

Partial Soundness for Approximated Service Basis under Restriction. The partial soundness is not guaranteed for our verification algorithm when approximated service basis is used. However, when the initial services are specified in some restricted form, even though the approximated service basis is over-approximated, the partial soundness of our query searching algorithm can be proved as well. One possible restriction is that for any two timestamps t and t' in every initial service with $\mathcal{B}$, $d(\mathcal{B}', t, t')$ is required to be no less than 0. If the ceiling is set to be larger than $max(\mathcal{B}_q) + 1$ where $\mathcal{B}_q$ is the timing constraints of the query, we prove the partial soundness of our verification algorithm as follows. First, we prove that, under this restriction, for any service S in the approximated service basis, we have a corresponding service S' in the guided service basis such that $S = \alpha^U_{approx}(S')$. Second, when the contradiction instance set $\mathcal{R}_f$ is not empty for the approximated service basis, we prove the existence of a corresponding attack instance in the guided service basis. According to the Theorem 1, the attack found in the guided service basis is guaranteed to be valid. So the protocol indeed has an attack and the partial soundness for the approximated service basis under the restriction is then proved.

Lemma 1. *Given an initial service set $\mathcal{R}_{init}$ and a ceiling U. Every service in $\mathcal{R}_{init}$ satisfies the restriction that for any two timestamps t and t' in the service with $\mathcal{B}$, $d(\mathcal{B}', t, t')$ is no less than 0. We have $\forall S \in \beta(\alpha^U_{approx}, \mathcal{R}_{init})$, $\exists S' \in \beta(\alpha_{guided}, \mathcal{R}_{init})$ such that $S = \alpha^U_{approx}(S')$.*

Theorem 5. ***Partial Soundness under Restriction.*** *Assume $\mathcal{R}$ is $\mathcal{R}_{approx}$. Every service in $\mathcal{R}_{init}$ satisfies the restriction that for any two timestamps t and t' in the service with $\mathcal{B}$, $d(\mathcal{B}', t, t')$ is no less than 0. If the ceiling U is set to be larger than $max(\mathcal{B}_q)+1$ where $\mathcal{B}_q$ is the timing constraints of the query, $\mathcal{R} = \beta(\alpha^U_{approx}, \mathcal{R}_{init})$. Let $\mathcal{Q}$ be a query of $f' \leftarrow\!\!\dashv [\, \mathcal{B}' \,] \vdash H'$ and $S_q = H' \dashv [\, \mathcal{B}' \,] \vdash\!\!\rightarrow f'$. There exists S derivable from $\mathcal{R}_{init}$ such that S is a contradiction to $\mathcal{Q}$ if there exists $S' \in \mathcal{R}$ such that the output of S' is an accept event and $S_q \not\Rightarrow S'$.*

Due to the limitation of space, the detailed proofs are available in Appendix B. We also indicate whether this restriction is applicable to the experiments evaluated in Section 4.

Remarks. Given a protocol with a valid attack, there should exist a derivation tree for that attack. Since we do not bound the number of events presented in a derivation tree (a composite service), we effectively deal with an unbounded number of sessions. The reason why our algorithm could work (i.e., terminate with correct result) is mainly because of two reasons. First, different from the explicit attack searching, we do not actively instantiate the variables in the services. So it becomes possible to represent the infinite adversary behaviors with a finite number of services. Second, we made a reasonable assumption in this work such that different nonces have different values. If the same nonce is generated in two sessions, those two sessions should be the same. Thus we merge them during the verification. As a consequence, even though we do not abstract the nonces used in the protocol as ProVerif does, this assumption could help us to find inconsistency in the service and remove the invalid ones from the service basis.

4 Implementation and Experiments

The flexibility and expressiveness of our service framework make it suitable for specifying and verifying timed security protocols, for instance, timed authentication protocols and distance bound protocols, etc. We have implemented our verifier TAuth in C++ with about 8K LoC. All the experiments shown in this section are conducted under Mac OS X 10.9.1 with 2.3 GHz Intel Core i5 and 16G 1333MHz DDR3. The TAuth verifier and the models shown in this section are available in [1].

We summarize some implementation choices in TAuth below. First, the timing constraints in the service are represented by Difference Bound Matrices (DBMs) [5]. Since timestamps are unified and new timestamps are introduced in the service composition, we use unique identifiers to distinguish the timestamps generated in the system so that different timestamps have different identifiers among services. Second, events in a service are merged when the encoded fresh nonces are evaluated to a same value. The reason is that the value of nonces generated in the session should be random, so different fresh nonces should have different values. For instance, if the session key k is initiated in the *init* event, $init(A[], R_1, [k])$ and $init(I, R_2, [k])$ should be merged and the substitution $\{A[]/I, R_1/R_2\}$ should be applied to the service. If such events cannot be merged, the service is invalid. Third, we check the query contradiction on the fly when new services are composed. Whenever we find a contradiction, we stop the verification process and report the security flaw. This optimization can potentially give the early termination to the verification process when the protocol has security flaws.

Several different types of security protocols are analyzed in our experiments. In the experiments, *all* the protocols are proved or dis-proved in a short time as summarized in Table 2. For some protocols, the restriction mentioned in the Section 3.2 is applicable, so that the attack is guaranteed to be correct whenever it can be found, which is indicated in the table. Notice that, even though some protocols do not satisfy the restriction, all the attacks found in the experiments are valid. First, untimed protocols such as Needham-Schroeder series and SKEME are analyzed with TAuth. We use these protocols to show that TAuth can work with untimed protocols. Additionally, timed protocols like CCITT series are also checked by TAuth. However, the attacks found in these protocols are untimed. Furthermore, timed authentication protocols like the WMF series and the NS PK Lowe NC Time are correctly analyzed as well. We use these protocols to demonstrate that our approach can work with timed protocols and find timed attacks. Specifically, in the NS PK Lowe Nb Compromise version, the nonces generated by the responder in the protocol could be compromised [15], so the adversary could perform attacks to the protocol. Denning and Sacco [15] proposed a way to fix these security flaws by checking the timestamps. In the NS PK Lowe NC Time version, we assume that extra time is needed for the nonce compromise, so that freshness checking for the messages could ensure the authentication is attack-free. Notice that the service approximation only works for WMF Lowe version [21] in our experiment, that is a version of WMF fixed by Gavin Lowe, because it is the only protocol that cannot be early terminated by the on-the-fly algorithm (it is attack-free) and its timing constraints involve infinite expansion.

Moreover, we successfully analyze two distance bounding protocols, that are Auth Range [7, 9] and Ultrasound Dist Bound [29]. In the Auth Range protocol, the prover

Protocol	$\mathcal{R}_{guided}$			$\mathcal{R}_{approx}$			
	$\sharp\mathcal{R}$[a]	Result	Time	$\sharp\mathcal{R}$	Result	Restriction[b]	Time
Wide Mouthed Frog [8]	26	Attack [21]	3ms	26	Attack	SAT	4ms
Wide Mouthed Frog c [14]	19	Secure	3ms	19	Secure	SAT	3ms
Wide Mouthed Frog Lowe [21]	-	-	-	32	Secure	SAT	8ms
CCITT X.509(1) [10]	35	Attack [2]	4ms	35	Attack	SAT	3ms
CCITT X.509(1c) [2]	45	Secure	7ms	45	Secure	SAT	7ms
CCITT X.509(3) [10]	111	Attack [8]	52ms	111	Attack	SAT	51ms
CCITT X.509(3) BAN [8]	106	Secure	74ms	106	Secure	SAT	70ms
NS PK [26]	50	Attack [20]	6ms	50	Attack	SAT	6ms
NS PK Lowe [20]	51	Secure	8ms	51	Secure	SAT	9ms
NS PK Lowe Na Compromise [15]	51	Secure	8ms	51	Secure	SAT	8ms
NS PK Lowe Nb Compromise [15]	42	Attack [15]	3ms	42	Attack	SAT	3ms
NS PK Lowe NC Time [15]	48	Secure	10ms	48	Secure	UNSAT	10ms
SKEME [19]	77	Secure	73m	77	Secure	SAT	74ms
Auth Range [7, 9]	17	Secure	2ms	17	Secure	UNSAT	1ms
Ultrasound Dist Bound [29]	35	Attack [30]	2ms	35	Attack	UNSAT	2ms

Table 2. Verification results for timed authentication protocols

[a] Overall service number generated in the verification.

[b] Whether the restriction is satisfied for the initial service specification. When it is satisfied, the attack found is guaranteed to be valid.

wants to convince the verifier that he is within a pre-agreed distance with the verifier. For instance, in a keyless entry system frequently adopted by cars, the prover is the remote key and verifier is the car. In the Auth Range protocol, it is assumed that the prover is honest and nothing can travel faster than light, so they could securely use the travel time of radio signals to measure the distance. In the Ultrasound Dist Bound protocol which has the same application scenario as the Auth Range protocol, the verifier uses radio signals to send requests while the prover uses ultrasound to return the answers. Since ultrasound travels much slower than radio and other processing time is negligible, the travel time of ultrasound dominates the whole protocol execution time. However, this protocol does not require the prover to be honest, so the prover can send his answer by either radio or ultrasound to others. When the adversary has a cooperator near the verifier, he can send the answer to the cooperator by radio and ask the cooperator to forward the answer by ultrasound to the verifier. As a consequence, the verifier can be convinced that the prover is within the distance even though the prover is not.

Finally, we compare our tool TAuth with other successful untimed protocol verifiers, i.e., ProVerif [6] and Scyther [13]. The Needham Schroeder public key authentication protocols except for its timed variant are chosen for the comparison as timestamps are absent in these protocols. The comparison results are summarized in the Table 3. It can be seen that TAuth is almost as fast as ProVerif. TAuth is slightly slower mainly due to overhead on handling timing constraints. Thanks to the on-the-fly algorithm, TAuth is faster than ProVerif in finding the attack for the Lowe Nb Compromise version. Furthermore, TAuth is much faster than Scyther. Notice that Scyther could only verify

Protocol	Result	TAuth	ProVerif	Scyther
NS PK	Attack	6ms	6ms	200ms
NS PK Lowe	Secure	8ms	5ms	177ms
NS PK Lowe Na Compromise	Secure	8ms	5ms	170ms
NS PK Lowe Nb Compromise	Attack	3ms	5ms	31ms

Table 3. Comparison with other untimed protocol verifiers.

the Lowe version and Lowe Na Compromise version with a bounded number of sessions while TAuth proves for infinitely many sessions.

5 Conclusions and Discussions

We present a service framework which can automatically verify the timed authentication protocols with an unbounded number of sessions. The partial correctness of our approach have been formally proved in this work. The experiment results for four different types of scenarios show that our framework is efficient and effective to verify a large range of timed security protocols. Even though we only check timed authentication properties for security protocols in this work, our framework could be easily extended to secrecy checking with timing constraints.

For future works, a throughout study on the termination of the algorithm would be very interesting. Since the problem of verifying security protocols is undecidable in general, we cannot guarantee the termination of our algorithm, but identifying the terminable scenario for practical security protocol could help the general adoption of our techniques. Our approach is inspired by the method used in ProVerif [6]. As is discussed in Section 3, TAuth is as terminable as ProVerif when the service approximation is used. However, the over-approximation also introduces false alarms. In order to remove the false alarms, as is discussed in Section 3, we can apply some restriction to the specification so that the found attacks are guaranteed to be valid. However, the restriction mentioned previously is quite restrictive because network latency, brute force attack, etc. cannot be specified under that restriction. Hence, how to restrict the specification in a practical way is another interesting future work direction.

Acknowledgements. The authors are grateful to Jun Pang, Jingyi Wang and the anonymous reviewers for valuable comments on earlier versions of this paper. This project is partially supported by project IGDSi1305012 from SUTD.

References

1. TAuth tool and experiment models. Available at `http://www.comp.nus.edu.sg/~li-li/r/tauth.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. Anderson and R. Needham. Programming satan's computer. In *Computer Science Today*, volume 1000, pages 426–440. Springer Berlin Heidelberg, 1995.

4. D. A. Basin, S. Capkun, P. Schaller, and B. Schmidt. Formal reasoning about physical properties of security protocols. *ACM Trans. Inf. Syst. Secur.*, 14(2):16, 2011.
5. R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
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. S. Capkun and J.-P. Hubaux. Secure positioning in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2):221–232, 2006.
10. CCITT. The directory authentication framework - Version 7, 1987. Draft Recommendation X.509.
11. 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 Computer Society, 1999.
12. R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed model checking of security protocols. In *FMSE*, pages 23–32. ACM, 2004.
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. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, 1981.
16. N. Evans and S. Schneider. Analysing time dependent security properties in csp using pvs. In *ESORICS*, pages 222–237. Springer, 2000.
17. A. Francillon, B. Danev, and S. Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *NDSS*. The Internet Society, 2011.
18. G. Jakubowska and W. Penczek. Is your security protocol on time? In *FSEN*, pages 65–80. Springer, 2007.
19. H. Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. In *NDSS*, pages 114–127. IEEE Computer Society, 1996.
20. G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.
21. G. Lowe. A family of attacks upon authentication protocols. Technical report, Department of Mathematics and Computer Science, University of Leicester, 1997.
22. G. Lowe. A hierarchy of authentication specification. In *CSFW*, pages 31–44. IEEE Computer Society, 1997.
23. G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1-2):53–84, 1998.
24. G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(1):89–146, 1999.
25. 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.
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. K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Capkun. Proximity-based access control for implantable medical devices. In *CCS*, pages 410–419. ACM, 2009.
28. A. W. Roscoe and P. J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(1):147–190, 1999.
29. N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *Workshop on Wireless Security*, pages 1–10. ACM, 2003.

30. S. Sedighpour, S. Capkun, S. Ganeriwal, and M. B. Srivastava. Implementation of attacks on ultrasonic ranging systems (demo). In *SenSys*, page 312. ACM, 2005.
31. D. X. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1-2):47–74, 2001.

Appendix A. Proofs

Lemma 2. *If $S_o \circ_f S'_o$ is defined, $S_t \Rightarrow S_o$ and $S'_t \Rightarrow S'_o$, then either there exists f' such that $S_t \circ_{f'} S'_t$ is defined and $S_t \circ_{f'} S'_t \Rightarrow S_o \circ_f S'_o$, or $S'_t \Rightarrow S_o \circ_f S'_o$.*

Proof. Let $S_o = H_o \dashv\!\![\ \mathcal{B}_o\ ]\!\!\mapsto f_o$, $S'_o = H'_o \dashv\!\![\ \mathcal{B}'_o\ ]\!\!\mapsto f'_o$, $S_t = H_t \dashv\!\![\ \mathcal{B}_t\ ]\!\!\mapsto f_t$, $S'_t = H'_t \dashv\!\![\ \mathcal{B}'_t\ ]\!\!\mapsto f'_t$. There should exist a substitution σ such that $\sigma f_t = f_o$, $\sigma H_t \subseteq H_o$, $\sigma f'_t = f'_o$, $\sigma H'_t \subseteq H'_o$, $\sigma\mathcal{B}_t \supseteq \mathcal{B}_o$, $\sigma\mathcal{B}'_t \supseteq \mathcal{B}'_o$. Assume $S_o \circ_f S'_o = \mathit{clear}(\sigma'(H_o \cup (H'_o - f))) \dashv\!\![\ \mathit{sim}(\sigma'\mathcal{B}_o \cap \sigma'\mathcal{B}'_o)\ ]\!\!\mapsto \sigma' f'_o$. We discuss the two cases as follows.

First case. Suppose $\exists f' \in H'_t$ such that $\sigma f' = f$. Since $S_o \circ_f S'_o$ is defined. f and f_o are unifiable. Let σ' be the most general unifier, $\sigma'\sigma f' = \sigma'\sigma f_t$, then f' and f_t are unifiable, therefore $S_t \circ_{f'} S'_t$ is defined. Let σ_t be the most general unifier, then $\exists\sigma'_t$ such that $\sigma'\sigma = \sigma'_t\sigma_t$. We have $S_t \circ_{f'} S'_t = \mathit{clear}(\sigma_t(H_t \cup (H'_t - f'))) \dashv\!\![\ \mathit{sim}(\sigma_t\mathcal{B}_t \cap \sigma_t\mathcal{B}'_t)\ ]\!\!\mapsto \sigma_t f'_t$. Since $\sigma'_t\sigma_t(H_t \cap (H'_t - f')) = \sigma'\sigma(H_t \cup (H'_t - f')) \subseteq \sigma'(H_o \cup (H'_o - f))$, $\sigma'_t\sigma_t f'_t = \sigma'\sigma f'_t = \sigma' f'_o$ and $\sigma'_t \mathit{sim}(\sigma_t\mathcal{B}_t \cap \sigma_t\mathcal{B}'_t) = \mathit{sim}(\sigma'_t\sigma_t\mathcal{B}_t \cap \sigma'_t\sigma_t\mathcal{B}'_t) = \mathit{sim}(\sigma'\sigma\mathcal{B}_t \cap \sigma'\sigma\mathcal{B}'_t) \supseteq \mathit{sim}(\sigma'\mathcal{B}_o \cap \sigma'\mathcal{B}'_o)$, we have $S_t \circ_{f'} S'_t \Rightarrow S_o \circ_f S'_o$.

Second case. Since $\forall f' \in H'_t$ such that $\sigma f' \neq f$, we have $\sigma H'_t \subseteq H'_o - f$. Thus $\sigma'\sigma H'_t \subseteq \sigma'(H_o \cup (H'_o - f))$, $\sigma'\sigma\mathcal{B}'_t \supseteq \sigma'\mathcal{B}'_o \supseteq \sigma'\mathcal{B}_o \cap \sigma'\mathcal{B}'_o$ and $\sigma'\sigma f'_t = \sigma' f'_o$. Therefore $S'_t \Rightarrow S_o \circ_f S'_o$. □

Theorem 1. For any service S in the form of $H \dashv\!\![\ \mathcal{B}\ ]\!\!\mapsto f$ where $\forall p \in H : p \in \mathcal{V}$, S is derivable from $\mathcal{R}_{init}$ if and only if S is derivable from $\mathcal{R}_{guided}$.

Proof. **(only if)** Assume S is derivable from $\mathcal{R}_{init}$, then there exists a derivation tree T_i for S on $\mathcal{R}_{init}$. Since a service S is removed from the basis only if it is implied by another service S', we have $\forall S \in \mathcal{R}_{init}, \exists S' \in \mathcal{R}_v, S' \Rightarrow S$.

As a result, we should be able to replace all the labels of nodes in T_i with services in $\mathcal{R}_v$ and get a new derivation tree T_v. Because some of the services are filtered out from $\mathcal{R}_v$ to $\mathcal{R}_{guided}$ when their input facts do not all belong to $\mathcal{V}$, we further need to prove that the nodes in T_v can be composed together until a derivation tree T_{guided} is formed so that all the nodes in T_{guided} are labeled by services in $\mathcal{R}_{guided}$.

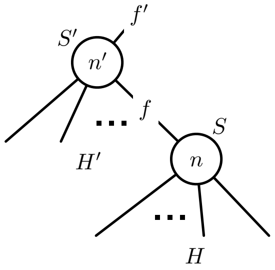
To continue our proof, we assume that there exists two nodes n and n' in T_v and they are linked by an edge f as shown in the right figure. We should have $S, S' \in \mathcal{R}_v$ such that $S \Rightarrow H \dashv\!\![\ \mathcal{B}\ ]\!\!\mapsto f$, $S' \Rightarrow H' \dashv\!\![\ \mathcal{B}'\ ]\!\!\mapsto f'$ and $f \in H'$. If $S_f = (H \dashv\!\![\ \mathcal{B}\ ]\!\!\mapsto f) \circ_f (H' \dashv\!\![\ \mathcal{B}'\ ]\!\!\mapsto f')$ is defined, according to Lemma 2, we could replace the two nodes with only one node in two different cases. In the first case, because $\mathcal{R}_v$ is the fixed-point of the service composition, there should exist $S'' \in \mathcal{R}_v$ such that $S'' \Rightarrow S_f$. In the second case, we can remove the node n and link its incoming links directly to the n', so that the new node n' is still implied by S'. We could continuously replace the nodes in the derivation tree until no node can be further processed and we denote the new tree as T_{guided}. For every node in T_{guided}, we prove the services labeled to the nodes are in $\mathcal{R}_{guided}$ as follows.

- For the leaves of the tree, their incoming edges are labeled by the facts in $\mathcal{V}$. So the leaves are labeled by services in $\mathcal{R}_{guided}$.
- For an inner node n' of the tree with all its children's service inputs in $\mathcal{V}$. Because n' cannot composed by its children, the inputs of the service labeled to n' should also be facts in $\mathcal{V}$. So the services labeled to all the inner nodes are in $\mathcal{R}_{guided}$.

As a consequence, all the nodes in T_{guided} are labeled by services in $\mathcal{R}_{guided}$, so S is derivable from $\mathcal{R}_{guided}$.

(if) For every service in $\mathcal{R}_v$, it should be composed from existing services, which is in turn composed from the initial service set. Thus all the services in $\mathcal{R}_v$ should be derivable from $\mathcal{R}_{init}$. In the meanwhile, $\mathcal{R}_{guided}$ does not introduce extra services except for existing services in $\mathcal{R}_v$, so $\forall S \in \mathcal{R}_{guided}$, S is derivable from $\mathcal{R}_{init}$. □

Theorem 2. Let U be the ceiling. For any service S in the form of $H \;{-}\!\!\![\; \mathcal{B} \;]\!\!\!\rightarrowtail f$ where $\forall f_{in} \in H : f_{in} \in \mathcal{V}$, if S is derivable from $\mathcal{R}_{init}$, S is also derivable from $\mathcal{R}_{approx}$.



Proof. The proof for this theorem is almost the same as the *only if* proof for Theorem 1. Since service approximation only expands the timing constraints, given an initial service set $\mathcal{R}_{init}$ and a ceiling U, we should have that $\forall S \in \beta(\alpha_{guided}, \mathcal{R}_{init})$, $\exists S' \in \beta(\alpha^{U}_{approx}, \mathcal{R}_{init})$, $S' \Rightarrow S$. Because the only if for Theorem 1 is already proved, we should also have that S is derivable from $\mathcal{R}_{approx}$ whenever S is derivable from $\mathcal{R}_{init}$. □

Theorem 3. Partial Soundness. Assume $\mathcal{R}$ is $\mathcal{R}_{guided}$. Let $\mathcal{Q}$ be a query of $f' \hookleftarrow\!\![\; \mathcal{B}' \;]\!\!- H'$ and $S_q = H' \;{-}\!\!\![\; \mathcal{B}' \;]\!\!\!\rightarrowtail f'$. There exists S derivable from $\mathcal{R}_{init}$ such that S is a contradiction to $\mathcal{Q}$ if there exists $S' \in \mathcal{R}$ such that the output of S' is an accept event and $S_q \not\Rightarrow S'$.

Proof. $[\mathcal{R} = \mathcal{R}_{guided}]$ Since $S' \in \mathcal{R}$ of which the output is an accept event and $S_q \not\Rightarrow S'$, according to the Definition 7, S' is a contradiction to the query $\mathcal{Q}$. On the other hand, according to Theorem 1, since $S' \in \mathcal{R}$, S' is derivable from $\mathcal{R}_{init}$. □

Theorem 4. Partial Completeness. Assume $\mathcal{R}$ is either $\mathcal{R}_{guided}$ or $\mathcal{R}_{approx}$. Let $\mathcal{Q}$ be a query of $f' \hookleftarrow\!\![\; \mathcal{B}' \;]\!\!- H'$ and $S_q = H' \;{-}\!\!\![\; \mathcal{B}' \;]\!\!\!\rightarrowtail f'$. There exists S derivable from $\mathcal{R}_{init}$ such that S is a contradiction to $\mathcal{Q}$ only if there exists $S' \in \mathcal{R}$ such that the output of S' is an accept event and $S_q \not\Rightarrow S'$.

Proof. [$\mathcal{R} = \mathcal{R}_{guided}$ *or* $\mathcal{R}_{approx}$] If S derivable from $\mathcal{R}_{init}$ is a contradiction to the $\mathcal{Q}$, according to Theorem 1 (Theorem 2 resp.), there is a derivation tree T with its nodes labeled by services in $\mathcal{R}_{guided}$ ($\mathcal{R}_{approx}$ resp.) for S. Suppose the root of T is labeled by $S_r \in \mathcal{R}_{guided}$ ($S_r \in \mathcal{R}_{approx}$ resp.), the output of S_r is an accept event, and the inputs of S_r are facts in $\mathcal{V}$. Because nodes in the derivation tree cannot be connected by events, the inputs corresponding to the edges connecting the children of root are singletons. If $S_q \Rightarrow S_r$, S_r implies the node and all of the input events in S_r is also in S, we have $S_q \Rightarrow S$, which conflicts the precondition $S_q \not\Rightarrow S$. Thus $S_q \not\Rightarrow S_r$. As S_r is in $\mathcal{R}$ such that the output of S_r is an accept event, the theorem is then proved. □

Appendix B. Partial Soundness under Restriction

Lemma 1. Given an initial service set $\mathcal{R}_{init}$ and a ceiling U. Every service in $\mathcal{R}_{init}$ satisfies the restriction that for any two timestamps t and t' in the service with $\mathcal{B}$, $d(\mathcal{B}', t, t')$ is no less than 0. We have $\forall S \in \beta(\alpha^U_{approx}, \mathcal{R}_{init})$, $\exists S' \in \beta(\alpha_{guided}, \mathcal{R}_{init})$ such that $S = \alpha^U_{approx}(S')$.

Proof. First, we need to prove the equation $\alpha^U_{approx}(\alpha^U_{approx}(S) \circ_f \alpha^U_{approx}(S')) = \alpha^U_{approx}(S \circ_f S')$ is hold when S and S' satisfy the restriction. Assume $\mathcal{B}$ is the timing constraint set for S, and $\mathcal{B}'$ for S', $\mathcal{B}_a$ for $\alpha^U_{approx}(S)$, $\mathcal{B}'_a$ for $\alpha^U_{approx}(S)'$ respectively. Given $d = d(\mathcal{B}, t, t')$, $d' = d(\mathcal{B}', t', t'')$, $d_a = d(\mathcal{B}_a, t, t')$ and $d'_a = d(\mathcal{B}'_a, t, t')$, we discuss different cases for d and d' as follows.

- If $d > U \wedge d' > U$, we have $d_a = \infty \wedge d'_a = \infty$, so $d + d' > U \wedge d_a + d'_a > U$.
- If $d > U \wedge 0 \leq d' \leq U$, we have $d_a = \infty \wedge d'_a = d'$, so $d + d' > U \wedge d_a + d'_a > U$.
- If $0 \leq d \leq U \wedge d' > U$, we have $d_a = d \wedge d'_a = \infty$, so $d + d' > U \wedge d_a + d'_a > U$.
- If $0 \leq d \leq U \wedge 0 \leq d' \leq U$, we have $d_a = d \wedge d'_a = d'$, so $d + d' = d_a + d'_a$.

Thus we have the timing constraints in $\alpha^U_{approx}(\alpha^U_{approx}(S) \circ_f \alpha^U_{approx}(S'))$ are the same as those in $\alpha^U_{approx}(S \circ_f S')$. Since the service approximation does not change the services inputs and output, the equation is thus valid. Given an approximated service in $\beta(\alpha^U_{approx}, \mathcal{R}_{init})$, there should exist a derivation tree labeled by services in $\mathcal{R}_{init}$ and the node is replaces by approximated services when two directly connected nodes are composed, according to the proof of Theorem 1. By repeatedly using the above equation during the node composition, we could delay the service approximation to a tree labeled by services in $\beta(\alpha_{guided}, \mathcal{R}_{init})$. Because the service approximation does not modify the service inputs and output, there should also be only one node in the tree. Hence we have $\forall S \in \beta(\alpha^U_{approx}, \mathcal{R}_{init})$, $\exists S' \in \beta(\alpha_{guided}, \mathcal{R}_{init})$ such that $S = \alpha^U_{approx}(S')$. □

Theorem 5. Partial Soundness under Restriction. Assume $\mathcal{R}$ is $\mathcal{R}_{approx}$. Every service in $\mathcal{R}_{init}$ satisfies the restriction that for any two timestamps t and t' in the service with $\mathcal{B}$, $d(\mathcal{B}', t, t')$ is no less than 0. If the ceiling U is set to be larger than $max(\mathcal{B}_q) + 1$ where $\mathcal{B}_q$ is the timing constraints of the query, $\mathcal{R} = \beta(\alpha^U_{approx}, \mathcal{R}_{init})$. Let $\mathcal{Q}$ be a query of $f' \leftarrow\!\dashv [\, \mathcal{B}' \,] \vdash H'$ and $S_q = H' \dashv [\, \mathcal{B}' \,] \vdash\!\rightarrow f'$. There exists S derivable from $\mathcal{R}_{init}$ such that S is a contradiction to $\mathcal{Q}$ if there exists $S' \in \mathcal{R}$ such that the output of S' is an accept event and $S_q \not\Rightarrow S'$.

Proof. [$\mathcal{R} = \mathcal{R}_{approx}$] According to Lemma 1, there should exist a service S_g in $\beta(\alpha_{guided}, \mathcal{R}_{init})$ such that $S' = \alpha^{U}_{approx}(S_g)$. Since the ceiling U is set to be larger than $max(\mathcal{B}_q) + 1$, we have $S_q \not\Rightarrow S_g$ as well. Because of Theorem 3, we have S_g is derivable from $\mathcal{R}_{init}$ such that S_g is a contradiction to $\mathcal{Q}$. The theorem is thus proved. □

Appendix C. Algorithms

Algorithm 1: Service Basis Construction Algorithm

```
Input  : $\mathcal{R}_{init}$ - the initial service set
Input  : $\alpha$ - the service approximation function
Output: $\mathcal{R}$ - the generated service basis
Procedure $add$ ($S$, $\mathcal{R}$)
    for $S_b \in \mathcal{R}$ do
        if $S_b \Rightarrow S$ then return $\mathcal{R}$;
        if $S \Rightarrow S_b$ then $\mathcal{R} = \mathcal{R} - \{S_b\}$;
    return $\{S\} \cup \mathcal{R}$;
Algorithm
    $\mathcal{R} = \emptyset$;
    for $S \in \mathcal{R}_{init}$ do
        $\mathcal{R} = add(S, \mathcal{R})$;
    repeat
        select $S_l$, $S_r$ from $\mathcal{R}$ such that:
        $\forall f_{in} \in inputs(S_l) : f_{in} \in \mathcal{V} \wedge \exists f : f \not\in \mathcal{V} \wedge S_l \circ_f S_r$ is defined;
        $\mathcal{R} = add(\alpha(S_l \circ_f S_r), \mathcal{R})$;
    until fixed-point reached;
    $\mathcal{R}_v = \mathcal{R}$;              /* $\mathcal{R}_v$ is reserved for the proof */
    return $\{S | S \in \mathcal{R} \wedge \forall f_{in} \in inputs(S) : f_{in} \in \mathcal{V}\}$;        /* $\mathcal{R}$ */
```

Algorithm 2: Query Contradiction Searching

```
Input  : $\mathcal{R}$ is either $\mathcal{R}_{guided}$ or $\mathcal{R}_{approx}$
Input  : $\mathcal{Q}$ is the query of the form of $a \leftarrow\!\dashv [\, \mathcal{B} \,] \vdash i$
Output: $\mathcal{R}_f$ is the contradiction instances
Algorithm
    $S_q = i \dashv [\, \mathcal{B} \,] \mapsto a$; $\mathcal{R}_f = \emptyset$;
    for $S \in \mathcal{R}$ and the output of $S$ is an accept event do
        if $S_q \not\Rightarrow S$ then $\mathcal{R}_f = \{S\} \cup \mathcal{R}_f$;
    return $\mathcal{R}_f$;
```