Symbolic Timed Trace Equivalence

Vivek Nigam¹,², Carolyn Talcott³ and Abraão Aires Urquiza¹

¹ Federal University of Paraíba, Brazil, abraauc@gmail.com
² fortiss GmbH, Germany, nigam@fortiss.org
³ SRI International, USA, clt@csl.sri.com

Abstract. Intruders can infer properties of a system by measuring the time it takes for the system to respond to some request of a given protocol, that is, by exploiting time side channels. These properties may help intruders distinguish whether a system is a honeypot or concrete system helping them avoid defense mechanisms, or track a user among others violating his privacy. Observational and trace equivalence are technical machineries used for verifying whether two systems are distinguishable. Automating the check for trace equivalence suffers the state-space explosion problem. Symbolic verification is used to mitigate this problem allowing for the verification of relatively large systems. This paper introduces a novel definition of timed trace equivalence based on symbolic time constraints. Protocol verification problems can then be reduced to problems solvable by off-the-shelf SMT solvers. We implemented such machinery in Maude and carry out a number of experiments demonstrating the feasibility of our approach.

Catherine Meadows has been a constant inspiration to us and to our work. We are honoured to be associated with Cathy as a colleague and a friend.

1 Introduction

This paper is dedicated to Catherine Meadows. Protocol security verification has emerged in the past decades as an exciting research field in which Meadows has had a key role. Meadows has been influential in the development of mathematical theories, formal languages, and tools for protocol analysis. Her technical expertise combined with deep insights into security issues has enabled her to successfully apply both formal and informal logical analysis to diverse aspects of computer security. Of particular importance is Meadow’s work formalizing and analyzing protocols and standards widely used in practice, leading to new insights into and improvements of the specifications and implementations. Examples include: the Internet Key Exchange protocol [33]; the RSA Laboratories Public Key Standards PKCS#11, a widely used application API to crypto libraries [24]; and fully automatic analysis of YubiKey and the YubiHSM hardware security module [23]. Meadows’ careful formalizations and insights into attack models has been particularly inspiring in our work, especially analysis of Distance Bounding Protocols with application to complexity reduction and collusion analysis [35]; Denial of Service [34]; and most related to this paper is recent work resulting in a formal definition of protocol indistinguishability and a method to check using Maude NPA [41].
Much of Meadows’ recent work has been undertaken using the Maude NPA tool [19] which is implemented in the Maude rewriting logic system. Maude NPA builds on ideas developed for NPA [32], an earlier tool developed by Meadows, and supports reasoning about cryptographic algorithms subject to equational theories. Meadows has been instrumental in guiding the extensions of Maude to support unification and narrowing in the presence of a rich collection of crypto primitives.

One important aspect that has been foreseen by Meadows in previous papers, but not yet fully formalized/investigated are attacks using time side channels. Time side channels can be exploited by intruders in order to infer properties of systems, helping them avoid defense mechanisms, and track users, violating their privacy. For example [27], honeypots are normally used for attracting intruders in order to defend real systems from their attacks. However, as honeypots run on virtual machines whereas normal client systems usually do not, it takes longer for a honeypot to respond to some protocol requests. This information can be used by the attacker to determine which servers are real and which are honeypots. For another example [11], passports using RFID mechanisms have been shown to be vulnerable to privacy attacks. An intruder can track a particular’s passport by replaying messages of previous sessions and measuring response times.

The formal verification of whether an intruder can infer such properties is different from usual reachability based properties, such as secrecy, authentication and other correspondence properties. In the verification of reachability properties, one searches for a trace that exhibits the flaw, e.g., the intruder learning a secret. In attacks such as the ones described above, one searches instead for behaviors that can distinguish two systems, e.g., a behavior that can be observed when interacting with one system, but that cannot be observed when interacting with the other system. That is, to check whether the systems are observationally distinguishable. This requires reasoning over sets of traces.

Various notions of trace equivalence\(^4\) have been proposed in the programming languages community as well as in concurrent systems [2, 7, 26, 36] using, for example, logical relations and bisimulation. Trace equivalence has also been proposed for protocol verification notably the work of Cortier and Delaune [14]. A number of properties, e.g., unlinkability and anonymity [3], have been reduced to the problem of trace equivalence. As protocol verification involves infinite domains, the use of symbolic methods has been essential for the success of such approaches.

The contribution of this paper is three-fold:

- **Symbolic Timed Trace Equivalence:** We propose a novel definition of timed equivalence over timed protocol instances [39]. Timing information, e.g., duration of computation, is treated symbolically and can be specified in the form of time constraints relating multiple time symbols, e.g., \(t_1 \geq t_2 + 10\);

- **SMT Solvers for proving Timed Trace Equivalence:** SMT solvers are used in two different ways. We specify the operational semantics of timed protocols using Rewriting Modulo SMT [40], reducing considerably the search space needed to enumerate these traces.

---

\(^4\)and more generally, observational equivalence
The second application of SMT-Solvers is on the proof of timed trace equivalence, namely, to check whether the timing of observations can be matched. This check involves the checking for the satisfiability of $\exists\forall$ formulas [18].

The use of general SMT solvers means that our implementations can immediately profit from improvements made to these solvers. Also, as SMT solvers have large communities using them, we have some confidence on the soundness of the implementations.

**Implementation:** Relying on the Maude [12] support for Rewriting Modulo SMT using the SMT-solvers CVC4 [4] or Yices [18], we implemented in Maude the machinery necessary for enumerating symbolic traces. However, as checking for the satisfiability of $\exists\forall$ formulas [18] is not supported by Maude, we integrate our Maude machinery with the SMT solver Yices [18]. We carry out some proof-of-concept experiments demonstrating the feasibility of our approach.

This paper only considers the case of bounded protocol sessions. The case of unbounded protocol sessions is left to future work.

Section 2 describes some motivating examples of how an intruder can use time side channels for his benefit. We introduce the basic symbolic language and the timed protocol language in Section 3. Section 4 gives the operation semantics of the timed protocol language. Section 5 introduces symbolic timed trace equivalence describing how to prove this property. Section 6 describes our implementation architecture and the experiments carried out. Finally, in Section 7, we conclude by commenting on related and future work. Finally, missing proofs and additional material can be found in the accompanying technical report [38].

## 2 Examples

We discuss some motivating examples illustrating how intruders can exploit time side channels of protocols.

**Red Pill** Our first example is taken from [27]. The attack is based on the concept of *red pills*. As honeypots trying to lure attackers normally run on virtual machines, determining if a system is running on a virtual machine or not gives an attacker one means to avoid honeypots [27]. The system running in a virtual machine or a concrete machine follows exactly the same protocol.

When an application connects to the malicious server, the server first sends a *baseline* request followed by a *differential* request. The time to respond to the baseline request is the same whether running in a virtual machine or not and is used for calibration. The time to respond to the differential request is longer when executed in a virtual machine. When not taking time into account, the set of traces for this exchange is the same whether the application is running on a virtual machine or not. However, if we also consider the time to respond to the two requests, the timed traces of applications running on virtual machines can be distinguished from those of applications running on native hardware.

**Passport RFID** Our second example comes from work of Chothia and Smirnov [11] investigating the security of *e-passports*. These passports contain an RFID tag that, when powered, broadcasts information intended for passport readers. Chothia and Smirnov
identified a flaw in one of the passport protocols that makes it possible to trace the movements of a particular passport, without having to break the passports cryptographic key. In particular, if the attacker records one session between the passport and a legitimate reader, one of the recorded messages can be replayed to distinguish that passport from other passports. Assuming that the target carried their passport on them, an attacker could place a device in a doorway that would detect when the target entered or left a building. In the protocol, the passport receives an encryption and a mac verifying the integrity of the encryption. The protocol first checks the mac, and reports an error if the check fails. If the mac check succeeds, it checks the encryption. This will fail if the encryption isn’t fresh. When the recorded encryption, mac pair is replayed to the recorded passport, the mac check will succeed but the encryption check will fail, while the mac check will fail when carried out by any other passport as it requires a key unique to the passport. The time to failure is significantly longer for the targeted passport than for others, since only the mac check is needed and it is faster.

**Anonymous Protocol** Abadi and Fournet [1] proposed an anonymous group protocol where members of a group can communicate with each other without revealing that they belong to the same group. A member of a group broadcasts a message, \( m \), encrypted with the shared group key. Whenever a member of a group receives this message, it is able to decrypt the message and then check whether the sender indeed belongs to the group and if the message is directed to him. In this case, the receiver broadcasts an encrypted response \( m' \).

Whenever a player that is not member of the group receives the message \( m \), it does not simply drop the message, but sends a decoy message with the same shape as if he belongs to the group, i.e., in the same shape as \( m' \). In this way, other participants and outsiders cannot determine whether two players belong to the same group or not.

However, as argued in [13], by measuring the time when a response is issued, an intruder can determine whether two players belong to the same group. This is because decrypting and generating a response take longer than just sending a decoy message.

### 3 Timed Protocols

We use a basic message term language which contains the usual cryptographic operators such as encryption, nonces, and tuples, augmented with time constraints. Trace and observational equivalence involving terms richer than the ones we use has been subject of a number of works. Here we introduce enough to write our examples. As the specification of timing aspects of protocols are orthogonal to the term language, extensions to the term language, such as constructs for hash, MAC, fresh keys, etc., can be made without affecting our main results.

The term language is defined by the following grammar. We assume given countable sets for text constants, \( T \), player names, \( P \), nonces, \( N \), symmetric keys, \( K \), symbols, \( \text{Syms} \), and variables, \( V \), where \( T, P, N, K, \text{Syms} \) and \( V \) are disjoint. Below \( v_p \), repre-
sents a variable of sort player.

**Basic Constants:**
- \( c := t \in T \) Text Constants
- \( p \in P \) Player Names
- \( n \in N \) Nonces

**Keys:**
- \( k := \text{sym} \in \mathcal{K} \) Symmetric keys
- \( \text{pk}(p) \mid \text{sk}(v_p) \) Public key of a player
- \( \text{sk}(p) \mid \text{sk}(v_p) \) Secret key of a player

**Symbols:**
- \( \text{sym} := \mid \text{sym} \in \text{Syms} \) Symbol

**Terms:**
- \( m := c \) Basic constants
- \( k \) Keys
- \( v \in \mathcal{V} \) Variables
- \( \text{sym} \in \text{Syms} \) Symbols
- \( \text{e}(m,k) \) Encrypted term
- \( \langle m_1, \ldots, m_n \rangle \) Tuples

A term is **ground** if it does not contain any occurrence of variables and symbols. A term is **symbolic** if it does not contain any occurrence of variables, but it may contain occurrences of symbols. \( ms, ms_1, ms_2, \ldots \) will range over symbolic terms.

*Time* Assume a time signature, \( \Xi \), containing a set of numbers, \( r_1, r_2, \ldots \), a set of time variables, \( tt_1, tt_2, \ldots \), including the special variable \( \text{cur} \), and a set of pre-defined function symbols, including, \( +, -, \times, /, \text{floor}, \text{ceiling} \).

*Time Expressions* are constructed inductively from numbers and variables by applying function symbols to time expressions. For example, \( \text{ceiling}((2 + tt + \text{cur})/10) \) is a Time Expression. The symbols \( tr_1, tr_2, \ldots \) range over Time Expressions. The time variable \( \text{cur} \) is a keyword in our protocol specification language denoting the current global time. We do not constrain the set of numbers and function symbols in \( \Xi \), but, in practice, we allow only the signatures supported by the SMT solver used. We assume that time expressions are disjoint from message terms.

**Definition 1 (Symbolic Time Constraints).** Let \( \Sigma \) be a time signature. The set of **symbolic time constraints** is constructed using time expressions as follows: Let \( tr_1, tr_2 \) be time expressions, then

\[
tr_1 = tr_2, \quad tr_1 \geq tr_2, \quad tr_1 > tr_2, \quad tr_1 < tr_2, \quad \text{and} \quad tr_1 \leq tr_2
\]

are symbolic time constraints.

For example, \( \text{cur} + 10 < \text{floor}(tt - 5) \) is a time constraint. \( tc, tc_1, tc_2, \ldots \) will range over time constraints.

Intuitively, given a set of time constraints \( TC \), each of its models with concrete instantiations for the time variables corresponds to a particular scenario. This means that one single set of time constraints denotes a possibly infinite number of concrete scenarios. For example, the set of constraints \( \{tt_1 \leq 2, tt_2 \geq 1 + tt_1\} \) has an infinite number of models, e.g., \( \{tt_1 \mapsto 1.9, tt_2 \mapsto 3.1415\} \).

Finally, SMT-solvers, such as CVC4 [4] and Yices [18], can check for the satisfiability of a set of time constraints.
3.1 Timed Protocol Language

The language used to specify a timed cryptographic protocol, introduced in our previous work [39], has the standard constructions, such as the creation of fresh values, sending and receiving messages, and “if then else” constructors, each annotated with time constraints.

Definition 2 (Timed Protocols). A timed protocol consists of a set of timed protocol roles. The set of Timed Protocols Roles, \( \mathcal{P}_t \), is generated by the following grammar:

\[
\begin{align*}
\text{nil} & \quad \text{Empty Protocol} \\
| (\text{new } v \ # \ tc), \mathcal{P}_t & \quad \text{Fresh Constant} \\
| (+m \ # \ tc), \mathcal{P}_t & \quad \text{Timed Message Send} \\
| (-m \ # \ tc), \mathcal{P}_t & \quad \text{Timed Message Receive} \\
| (\text{if } (m_1 : = m_2 \ # \ tc \text{ then } \mathcal{P}_1 \text{ else } \mathcal{P}_2) & \quad \text{Timed Conditional}
\end{align*}
\]

Intuitively, \( \text{new} \) generates a fresh value binding it to the variable \( v \), \( (+m \ # \ tc) \) denotes sending the term \( m \) and \( (-m \ # \ tc) \) denotes receiving a term matching \( m \). For the term \( (\text{if } (m_1 : = m_2 \ # \ tc \text{ then } \mathcal{P}_1 \text{ else } \mathcal{P}_2) \), we assume that \( m_1 \) is ground when it is evaluated. Then if \( m_1 \) can be matched with \( m_2 \), that is, instantiate the variables in \( m_2 \) so that the resulting term is \( m_1 \), then the protocol proceeds to execute \( \mathcal{P}_1 \) and otherwise to execute \( \mathcal{P}_2 \). Moreover, the variables in \( m_2 \) are instantiated with the witnessing matching substitution in \( \mathcal{P}_1 \). We also assume that \( \mathcal{P}_2 \) does not contain variables in \( m_2 \). This is because the binding of these variables to concrete terms only happens when the condition is true. Finally, a command is only applicable if the associated constraint \( tc \) is satisfiable.

We elide the associated time constraint whenever \( tc \) is a tautology, i.e., it is always true. If we restrict the time constraints to be tautologies (say \( 1 = 1 \)), the timed protocol language can be considered as one of the usual security protocol specification languages. The following two examples illustrate this by specifying the traditional Needham-Schroeder protocol (suppressing trivially true time constraints).

Example 1. The Needham-Schroeder [37] protocol is specified as follows where \( X, Y, Z \) are variables:

\[
\begin{align*}
\text{Alice}(Z) & : = (\text{new } N_a), (+\text{e}(\langle N_a, alice, \text{pk}(Z) \rangle)), (-\text{e}(\langle N_a, Y, \text{pk}(alice) \rangle)), (+\text{e}(Y, \text{pk}(Z))) \\
\text{Bob} & : = (-\text{e}(\langle X, Z, \text{pk}(bob) \rangle)), (\text{new } N_b), (+\text{e}(\langle X, N_b, \text{pk}(Z) \rangle)), (-\text{e}(N_b, \text{pk}(bob)))
\end{align*}
\]

The “if then else” constructs and pattern matching allows to specify protocols with branching, as illustrated by the following example:

Example 2. Consider the following protocol role which is a modification of Alice’s role in the Needham-Schroeder’s protocol (Example 1):

\[
\begin{align*}
\text{Alice}(Z) & : = (\text{new } N_a), (+\text{e}(\langle N_a, alice, \text{pk}(Z) \rangle)), (-\nu), \\
& \quad \text{if } \nu : = \text{e}(\langle N_a, Y, \text{pk}(alice) \rangle) \text{ then } (+\text{e}(Y, \text{pk}(Z))) \text{ else } (+\text{error})
\end{align*}
\]

Here, Alice checks whether the received message \( \nu \) has the expected shape before proceeding. If it does not have this shape, then she sends an error message.
Time constraints can be used to specify timing aspects of security protocols. The following example illustrates how time constraints can specify the timing of a ping-pong message. It can be used to check the latency of a communication channel with a party. Similar constructs can be used to specify Distance-Bounding protocols [8].

Example 3. The following role specifies a ping-pong protocol where the response is only accepted within 4 time units:

\[(\text{new } v), (+v \# tt = \text{cur}), (-v \# \text{cur} \leq tt + 4)\]

It creates a fresh constant and sends it to the prover, remembering the current global time by assigning it to the time variable \(tt\). It only concludes if the response is received within 4 time units.

The next example illustrates how time constraints can be used to specify protocol decisions based on timing aspects.

Example 4. The following role modifies the ping-pong protocol:

\[(\text{new } v), (+v \# tt = \text{cur}), (-v_2 \# \text{tt}' = \text{cur}), (\text{if } (v := v_2 \# tt + 4 \geq \text{tt}') \text{ then } (+ok \# \text{cur} < \text{tt}' + 2) \text{ else } (+ko \# \text{cur} < \text{tt}' + 2))\]

It sends a nonce and receives a response. The protocol checks the response matches the nonce and whether it is received within 4 time units. If so, it responds \(ok\) and \(ko\) otherwise. Moreover, the response is sent within 2 time units after receiving the response.

Finally, as illustrated by the examples below, described in Section 2, time constraints can also be used to specify the duration of operations, such as checking whether some message is of a given form. In practice, the duration of these operations can be measured empirically to obtain a finer analysis of the protocol as done in [11].

Example 5 (Passport). Consider the following protocol role, taken from our previous work [39], which is the role of the passport used for identification.

\[(\text{new } v), (+v), (-\langle v_{enc}, v_{mac} \rangle \# \text{tt}_0 = \text{cur})\]

if \((v_{mac} := e(v_{enc}, k_M)) \# \text{tt}_1 = \text{tt}_0 + r_{mac}\) then

if \((v_{enc} := e(v, k_E)) \# \text{tt}_2 = \text{tt}_1 + r_{enc}\) then \((\text{done} \# \text{cur} = \text{tt}_2)\) else \((\text{error} \# \text{cur} = \text{tt}_2)\)

else \((\text{error} \# \text{cur} = \text{tt}_1)\)

This role creates a fresh value \(v\) and sends it. Then it is expecting a pair of two messages \(v_{mac}\) and \(v_{enc}\), which is received at \(\text{tt}_0\). It then checks whether the first component \(v_{mac}\) is of the form \(e(\text{tt}_0, k_M)\), i.e., it is the correct MAC. This operation takes \(r_{mac}\) time units. The time variable \(\text{tt}_1\) is equal to the time \(\text{tt}_0 + r_{mac}\), i.e., the time when the message was received plus the MAC check duration. If the MAC is not correct, an \(\text{error}\) message is sent at time \(\text{tt}_1\). Otherwise, if the first component, \(v_{mac}\), is as expected, the role checks whether the second component, \(v_{enc}\), is an encryption of the form \(e(v, k_E)\), which takes (a longer) time \(r_{enc}\). If so it sends the \(\text{done}\) message, otherwise the \(\text{error}\) message, both at time \(\text{tt}_2\) which is \(\text{tt}_1 + r_{enc}\).
Notice that instead of using concrete values \( r_{mac} \) and \( r_{enc} \) for the time of verifying the MAC and the encrypted terms, respectively, we could have specified intervals for these operations. For example, the time constraints \( t_0 + r_{mac^-} \leq t_1 \leq t_0 + r_{mac^+} \) express that it takes a time between \( r_{mac^-} \) and \( r_{mac^+} \) to check whether the MAC term is correctly formed.

**Example 6 (Red Pill Example).** We abstract the part of sending the baseline message, e.g., the messages that establish the connection to the server, and the part that sends the differential messages. We assume that it takes \( d_{Base} \) to complete the exchange of the baseline messages.

\[
(-\langle \text{baseline}_\text{req} \rangle \# t_0 = \text{cur}), (+\langle \text{baseline}_\text{done} \rangle \# \text{cur} = t_0 + d_{Base}), \\
(-\langle \text{diff}_\text{req} \rangle \# t_1 = \text{cur}), (+\langle \text{diff}_\text{done} \rangle \# \text{cur} = t_1 + d_{Appl})
\]

Then the part of the protocol that depends on the application starts. We abstract this part using the messages \( \text{diff}_\text{req} \) and \( \text{diff}_\text{done} \). If the application is running over a virtual machine, then \( d_{Appl} \) takes \( d_{Virtual} \) time units; otherwise \( d_{Appl} \) takes \( d_{Real} \) time units, where \( d_{Virtual} > d_{Real} \).

The intruder can distinguish whether an application is running over a virtual machine or not by measuring the time it takes to complete the exchange of \( \text{diff}_\text{req} \) and \( \text{diff}_\text{done} \) messages.

**Example 7 (Anonymous Protocol).** We specify (a simplified version of) the anonymous group protocol proposed by Abadi and Fournet for private authentication [1]. Whenever a broadcasted message is received by an agent, it checks whether it has been encrypted using his public key \( K_B \). If so, the agent learns the group key \( k_A \) and responds with a message encrypted with the group key. Otherwise, the agent sends a decoy message encrypted with a fresh key \( k_\nu \), only known to the agent.

\[
-(v), \text{if } v := \langle \text{hello}, e(\{\text{hello}, v_\nu, k_A\}, k_G) \rangle \# t_1 = \text{cur} + d_{Enc} \text{ then} \quad \\
\text{if } (k_G := KB \# t_2 = t_1 + d_{Chk} \text{ then} + ((\text{ack}, e(rsp, k_A))) \# \text{cur} = t_1 + d_{Crt} \text{ else} (\text{new } k_\nu), +((\text{ack}, e(\text{decoy}, k_\nu))) \# \text{cur} = t_1 \text{ else} (\text{new } k_\nu), +((\text{ack}, e(\text{decoy}, k_\nu))) \# \text{cur} = t_1
\]

Here \( d_{Enc} \), \( d_{Chk} \) and \( d_{Crt} \) are numbers specifying the time needed for, respectively, decrypting the received message, checking whether the key is the group key, and creating and sending the response message.

### 4 Operational Semantics for Timed Protocols

This section formalizes the operational semantics for configurations with a fixed number of timed protocol role instances. The operational semantics uses symbolic terms, where instead of instantiating variables with concrete terms, one uses symbolic terms, where each symbol represents a possibly infinite set of ground terms that satisfies suitable constraints. This simple idea has enabled the verification of security protocols, which have infinite search space on ground terms, but finite state space using symbolic terms [5, 10, 14].
In Subsection 4.1, we introduce the types of symbolic term constraints necessary for our examples. Methods needed for operations involving symbolic terms and solving constraints can be found in, for example, [14,38]. The specific methods and algorithms used in our Maude implementation are detailed in [38].

4.1 Symbolic Term Constraints

We will use two types of (capture avoiding) substitutions. **Variable substitutions** written $sb, sb_1, sb_2, \ldots$ which are maps from variables to symbolic terms $sb = [v_1 \mapsto ms_1, v_2 \mapsto ms_2, \ldots, v_n \mapsto ms_n]$. **Symbol substitutions** written $ssb, ssb_1, ssb_2, \ldots$ mapping symbols to symbolic terms $ssb = [sym_1 \mapsto ms_1, \ldots, sym_n \mapsto ms_n]$.

The operational semantics uses two forms of constraints: derivability constraints and equality constraints.

**Definition 3.** A derivability constraint has the form $dc(sym, S)$, where $S$ is a set of symbolic terms and $sym$ a symbol. This constraint denotes that $sym$ can be any (symbolic) term derived from $S$.

For example, the derivability constraint $dc(sym, \{alice, n_1, e(sym_2, sk(alice)), pk(bob)\})$ specifies that $sym$ may be instantiated by, e.g., the terms $\langle alice, e(sym_2, sk(alice)) \rangle$, $\langle alice, n_1 \rangle$, $e(alice, pk(bob))$, $e(alice, bob), pk(bob)$ and so on.

Notice that any $dc(sym, S)$ denotes a infinite number of symbolic terms due to the tupling closure. We will abuse notation and use $ms \in dc(sym, S)$ to denote that the symbolic term $ms$ is in the set of terms that can be derived from $S$. Moreover, we assume that for any given set of derivability constraints $DC$, there is at most one derivability constraint for any given $sym$.

We will need to determine whether a symbolic term $ms'$ can represent another symbolic term $ms$, written $ms \in DC(ms')$. Roughly, this means that $ms$ can be obtained by repeated substitution of a symbol, $sym$ in $ms'$ by one of its instantiations according to $dc(sym, S) \in DC$. For example, assume that $DC$ contains $dc(sym_1, \{t_1\}), dc(sym_2, \{t_2\})$, then $(t_1, t_2) \in DC((sym_1, sym_2))$. The technical report [38] describe an algorithm to decide $ms \in DC(ms')$.

**Definition 4.** A symbol substitution $ssb$ satisfies a set of derivability constraints $DC$, written, $ssb \vDash DC$, if for each $sym \mapsto ms \in ssb$, $ms \in DC(sym)$.

The following definition specifies the second type of symbolic term constraints called comparison constraints.
Definition 5. A comparison constraint is either an equality constraint $\text{eq}(ms_1, ms_2)$ or an inequality constraint $\text{neq}(ms_1, ms_2)$, where $ms_1, ms_2$ are symbolic terms.

A set $\mathcal{EQ}$ of comparison constraints is interpreted as a conjunction of constraints. We write $\mathcal{DC} \models \mathcal{EQ}$ to denote that $\mathcal{EQ}$ is satisfiable with respect to derivability constraints $\mathcal{DC}$. The set of equality constraints impacts the ground terms a symbolic term represents. For example, given $\text{dc}(\text{sym}_1, \{t_1\}), \text{dc}(\text{sym}_2, \{t_2\}) \in \mathcal{DC}$ and $\mathcal{EQ} = \{\text{eq}(\text{sym}_1, \text{sym}_2)\}$, the symbolic term $(\text{sym}_1, \text{sym}_2)$ may represent $(t_1, t_2)$ when considering only $\mathcal{DC}$, but not when considering both $\mathcal{DC}$ and $\mathcal{EQ}$, as it falsifies the constraint that $\text{sym}_1$ and $\text{sym}_2$ are equal, witnessed by the matching substitution $\theta = [\text{sym}_1 \mapsto t_1, \text{sym}_2 \mapsto t_2]$. We write $ms \in \mathcal{DC}(ms')_{\mathcal{EQ}}$ to denote when the symbolic term $ms'$ can represent the symbolic term $ms$ assuming the constraints $\mathcal{DC}$ and $\mathcal{EQ}$. Algorithms to decide $\mathcal{DC} \models \mathcal{EQ}$ and $ms \in \mathcal{DC}(ms')_{\mathcal{EQ}}$ are given in [38].

4.2 Symbolic Constraint Solving

For protocol verification, we assume a traditional Dolev-Yao intruder [15], i.e., he can construct messages from his knowledge by tupling and encrypting messages. However, he can only decrypt a message for which he possesses the inverse key.

Definition 6. An intruder knowledge $IK$ is a set of symbolic terms.

Suppose an honest player is ready to receive a message matching a term $m$, possibly containing variables. Rather than considering all possible ground instances of $m$ that the intruder could send, we consider a finite representation of this set, namely symbolic messages where the possible values of the symbols are constrained by derivability constraints. To compute this representation the intruder replaces variables with symbolic terms, possibly containing fresh symbols, and then constrains the symbols so that the allowed instances are exactly the terms matching $m$ that the intruder can derive from (allowed instantiations of) his current knowledge $IK$.

For example, consider the term $m = e(\langle v_1, \text{sym}, v_1, v_2 \rangle, k)$ (which is expected as input by an honest player). Here $v_1$ and $v_2$ are variables and $\text{sym}$ is constrained by derivability constraints $\mathcal{DC}$. We create two fresh symbols $\text{sym}_1$ and $\text{sym}_2$ for, respectively, the variables $v_1$ and $v_2$. Letting $\text{sb} = [v_1 \mapsto \text{sym}_1, v_2 \mapsto \text{sym}_2]$, we obtain $ms = \text{sb}[m] = e(\langle \text{sym}_1, \text{sym}, \text{sym}_1, \text{sym}_2 \rangle, k)$.

It remains to constrain the symbols so that $ms$ represents the ground terms matching instances of $m$ (given $\mathcal{DC}$) that the intruder can generate given $IK$.

The function $\text{sgen}$. We implemented a function called $\text{sgen}$ that enumerates representations of the required instances of $ms$. Each representation has the form $\{\text{ssb}, \mathcal{DC}_i\}$ where $\text{ssb}$ represents symbols that have been constrained to a single value and $\mathcal{DC}_i$ constrains the remaining symbols. In particular, $\text{sgen}(m, IK, \mathcal{DC})$ takes as input a term $m$, which is expected by the honest participant, the intruder knowledge $IK$ and the derivability constraints $\mathcal{DC}$ for the existing symbols. $\text{sgen}(m, IK, \mathcal{DC})$ then generates as output a pair:

$$\{\text{sb}, \{\text{ssb}_1, \mathcal{DC}_1\} \ldots \{\text{ssb}_k, \mathcal{DC}_k\}\}$$
where \( sb \) maps the variables of \( m \) to fresh symbols, and each \( \{ ssb, DC_i \} \) is a solution to the problem above for \( ms = sb[m] \). If \( k = 0 \), then there are no solutions, that is, the intruder is not able to generate a term which matches \( m \). We describe \( sgen \) informally and illustrate it with some examples. The full specification and proof can be found in [38]. A similar algorithm is also used by [14].

Intuitively, the function \( sgen \) constructs a solution by either matching \( m \) with a term in his knowledge \( IK \) or deriving \( m \) from terms in \( IK \). The following example illustrates the different cases involved:

**Example 8.** Consider the following cases for deriving the term \( m = e(\langle v, sym \rangle, k) \).

- **Case 1 (matching with a term in \( IK \)):** Assume:
  
  \[
  IK = \{ e(\langle na, sym_1 \rangle, k) \} \quad DC = dc(sym, \{ na, nc \}) dc(sym_1, S)
  \]

  Then the solution of \( sgen \) is:

  \[
  \{ sb, [sym_v \mapsto na, sym \mapsto sym_1, dc(sym_1, \{ na, nc \} \cap S)] \}
  \]

  where \( sb = [v \mapsto sym_v] \) and \( sym_v \) is a fresh symbol. Notice that since \( sym_v \) is mapped to a particular term \( \langle na \rangle \), no derivability constraint for it is generated. Additionally, notice that \( sym \) is constrained to be the same as \( sym_1 \). This causes the removal of the derivability constraint \( dc(sym, S) \);

- **Case 2 (constructing terms from \( IK \)):** Assume that \( k \in IK \) and \( IK \) has no encryption terms. Then the solution of \( sgen \) is:

  \[
  ([v \mapsto sym_v], [[], DC \cup \{ dc(sym, IK) \}])
  \]

  which corresponds to generating the term \( e([sym_v, sym], k) \). Moreover, \( sym_v \) can be any term derivable from the intruder’s knowledge.

- **Case 3 (No Solution):** Assume that \( IK = \{ e(\langle na, nb \rangle, k) \} \) and \( DC = dc(sym, \{ na, nc \}) \).

  Since \( sym \) cannot be instantiated to \( nb \), the intruder cannot use the term \( e(\langle na, nb \rangle, k) \).

### 4.3 Operational Semantics

The operational semantics of timed protocols is given by rules that rewrite configurations defined below:

**Definition 7.** A symbolic configuration has the form \( \langle P, IK, DC, EQ, TC \rangle \) of \( tG \), where

- \( P \) is a finite set of player roles of the form \( [n | pl | keys] \) composed of an identifier, \( n \), a protocol role \( pl \), and a set \( keys \), \( keys \), known to the player;
- \( IK \) is the intruder knowledge;
- \( DC \) is a set of derivability constraints;
- \( EQ \) is a set of comparison constraints;
- \( TC \) is a set of time constraints;
- \( tG \) is a time symbol representing global time.
The operational semantics of timed protocols is defined in Figure 1. The New rule replaces the (bound) variable \(v\) by a fresh nonce \(n'\). The Send rule sends a message \(ms\) which is then added to the intruder knowledge. The Receive rule expects a term of the form \(m\). The function \(s\text{gen}(m, IK, DC)\) returns the variable substitution \(sb\) and a set of solutions \([ssb, DC]\) \(css\). Each solution intuitively generates a different trace. We apply \(sb\) in the remaining of the program \(pl\) and apply the symbol substitution \(ssb\) to all symbols in the resulting configuration. This rule also has a proviso that the message \(ms = ssb[sb[m]]\) is encrypted with keys that can be decrypted by the honest participant. This is specified by the function \(is\text{Receivable}\). Finally, it also adds to the set of keys of the honest participant keys, the keys he can learn from the message \(ms\).

The rule \textbf{If-true} checks whether the terms \(ms_1\) and \(ms_2\) can be matched given the constraints \(DC\). This is done by the function \(s\text{genB}\) which uses \(s\text{gen}\). It first matches the structure of the terms in the matching problem \(ms_1 = ms_2\), where symbols are considered as variables. If they cannot be matched, then the empty solution set is returned. Otherwise, there is a witnessing match \(sym_1 \mapsto ms_1, \ldots, sym_n \mapsto ms_n\). It then calls \(s\text{gen}\) on the terms \((sym_1, \ldots, sym_n)\) and \((ms_1, \ldots, ms_n)\) returning its output. The rule \textbf{If-true} then adds the equality constraint to the set of comparison constraints.

Finally, the rule \textbf{If-false} adds the corresponding inequality constraint stating that \(ms_1\) and \(ms_2\) are not equal.

Example 9. Consider the Needham-Schroeder protocol in Example 1. Assume that the intruder initially only knows his secret key (and the guessables), \(IK_0 = \{sk(eve)\}\) and there are no symbols \(DC = \emptyset\). An execution of Alice’s protocol role is as follows. Alice creates a fresh constant \(N_a\) and sends the message \(e(\langle N_a, alice\rangle, pk(eve))\). At this point, the intruder knowledge is:

\[
IK_1 = IK_0 \cup \{N_a\}
\]

He now can send a message to Bob, namely \(e(\langle sym_1, sym_2\rangle, pk(bob))\) where \(sym_1, sym_2\) are fresh and constrained \(DC_1 = \{dc(sym_1, IK_1), dc(sym_2, IK_1)\}\). At this point, Bob creates a fresh value \(N_b\) and sends the message \(e(\langle sym_1, N_b\rangle, pk(sym_2))\). The intruder learns this message:

\[
IK_2 = IK_1 \cup \{e(\langle sym_1, N_b\rangle, pk(sym_2))\}
\]

Now, the intruder can fool alice by sending her a message of the form \(e(\langle N_a, Y\rangle, pk(alice))\). We create a fresh symbol \(sym_3\) for \(Y\) obtaining \(e(\langle N_a, sym_3\rangle, pk(alice))\) and attempt to generate this message from \(IK_2\) using \(s\text{gen}\). Indeed we can generate this message using \(e(\langle sym_1, N_b\rangle, pk(sym_2))\) \(\in IK_2\). This generates the \(ssb = [sym_1 \mapsto N_a, sym_2 \mapsto alice, sym_3 \mapsto N_b]\). This substitution is consistent with \(DC_1\). The protocol finishes by the intruder simply forwarding the message sent by alice to bob. Bob then thinks he is communicating with alice, but he is not.

Each rule has two general provisos. The first is that the resulting set of comparison constraints should be consistent.

The second, more interesting, condition is on the time symbols. Whenever a rule is applied, time constraints \(TC_1\) are added to the configuration’s constraint set. These time constraints are obtained by replacing cur in tc with \(tG_i\) together with the constraint
For any $i$ the comparison constraints and the set of time constraints should be satisfiable.

Operational semantics for basic protocols. In each rule $t_G$ is an instance of a rule in $R$ and $t_G$ is written $t_G \rightsquigarrow C$ where $C$ is a fresh nonce and $sb = [v \mapsto n']$

Let $\langle (n \mid (\text{new } v \# tc), pl \mid \text{keys}\rangle P, IK, DC, EQ, TC) @ t_G$

$\rightsquigarrow \langle (n \mid \text{sb}[pl] \mid \text{keys}\rangle P, IK, DC, EQ, TC, C_1) @ t_G$

where $n'$ is a fresh nonce and $\text{sb} = [v \mapsto n']$

Send: $\langle (n \mid (+ms \# tc), pl \mid \text{keys}\rangle P, IK, DC, EQ, TC) @ t_G$

$\rightsquigarrow \langle (n \mid pl \mid \text{keys}\rangle P, IK \cup \{ms\}, DC, EQ, TC, C_1) @ t_G$

Receive: $\langle (n \mid (-m \# tc), pl \mid \text{keys}\rangle P, IK, DC, EQ, TC) @ t_G$

$\rightsquigarrow \text{ssb}\{\langle n \mid \text{sb}[pl] \mid \text{addKeys}(ms, keys)\rangle P, IK, DC, EQ, TC, C_1\} @ t_G$

where $\text{sb}[\text{ssb}, DC_1] \text{css} = \text{sgen}(m, IK, DC)$ and $ms = \text{ssb}[\text{sb}[m]]$ and $isReceivable(ms, keys)$

If-true: $\langle (n \mid (\text{if } (ms_1 := ms_2 \# tc) \text{ then pl_1 else pl_2) \mid \text{keys}\rangle P, IK, DC, EQ, TC) @ t_G$

$\rightsquigarrow \text{ssb}\{\langle n \mid \text{sb}[pl_1] \mid \text{keys}\rangle P, IK, DC_1, EQ \cup \{eq(\text{sb}[ms_1], \text{sb}[ms_2])\}, TC_1\} @ t_G$

where $\text{sb}[\text{ssb}, DC_1] \text{css} = \text{sgenB}(ms_1 = ms_2, IK, DC)$

If-false: $\langle (n \mid (\text{if } (ms_1 := ms_2 \# tc) \text{ then pl_1 else pl_2) \mid \text{keys}\rangle P, IK, DC, EQ)\rangle$

$\rightsquigarrow \langle (n \mid pl_2 \mid \text{keys}\rangle P, IK, DC \cup DC , EQ \cup \{eq(m_1, ms_2)\})$

Fig. 1. Operational semantics for basic protocols. In each rule $t_G$ is the time constraint obtained by replacing $cur$ in $t_G$ by the global time $t_G$; and $C_1 = C \cup \{t_G \geq t_G_0, t_{G_1}\}$. The function $isReceivable$ checks whether the message $\text{ssb}[\text{sb}[m]]$ can be decrypted with the keys he has in $\text{keys}$. Every rule has the proviso that to be applicable, the set of comparison constraints and the set of time constraints should be satisfiable.

$t_G \geq t_G_0$ specifying that time can only advance. The rule is fired only if the resulting set of time constraints $(TC \cup TC_1)$ is consistent, which is done by calling an SMT solver. This way of specifying systems is called Rewriting Modulo SMT [40].

Definition 8. Let $R$ be the set of rules in Figure 1. A timed trace is a labeled sequence of transitions written $C_1 \stackrel{l_1}{\rightarrow} C_2 \stackrel{l_2}{\rightarrow} \cdots \stackrel{l_{n-1}}{\rightarrow} C_n$ such that for all $1 \leq i \leq n-1, C_i \rightarrow C_{i+1}$ is an instance of a rule in $R$ and $l_i$ is $+ms@t_G$ if it is an instance of Send rule sending term $ms$ at time $t_G$ and $ms@t_G$ if it is an instance of Receive rule receiving term $ms$ at time $t_G$, and $0$ otherwise.

The use of rewriting modulo SMT considerably reduces the search space. Timed protocols are infinite state systems, as time symbols can be instantiated by any (positive) real number. With the use of rewriting modulo SMT we simply have to accumulate constraints. Only traces with satisfiable sets of time constraints are allowed. Indeed, as we describe in Section 6, the number of traces is not only finite (as stated in the following Proposition), but very low (less than 40 traces).

Proposition 1. The set of traces starting from any configuration $C_0$ is finite.

Proposition 2. Let $\tau = C_1 \stackrel{l_1}{\rightarrow} C_2 \stackrel{l_2}{\rightarrow} \cdots \stackrel{l_{n-1}}{\rightarrow} C_n$ be a trace. For any configuration $C_i = (P_i, IK_i, DC_i, EQ_i, TC_i) @ t_G$, such that $1 \leq i \leq n$, the following holds:

- For any $i \leq j \leq n$, $DC_{\tau}(IK_i)_{eq} \subseteq DC_{\tau}(IK_j)_{eq}$, that is, the intruder knowledge can only increase.
1. Let $\text{sym}_k$ be a symbol new in some $C_i$, $k < i$ and let $\text{sym}_i$ be a symbol new in $C_i$. If $\text{dc}(\text{sym}_k, S_i), \text{dc}(\text{sym}_i, S_i) \in \mathcal{DC}_i$, then $S_k \subseteq S_i$. That is, symbols that are introduced by later transitions can be instantiated by more terms than symbols introduced at earlier transitions.

**Timed Intruders:** In fact, our implementation generalizes the machinery in this section by considering multiple timed intruders [29, 39]. As described in [29], the standard Dolev-Yao may not be suitable for the verification of Cyber-Physical Security Protocols where the physical properties of the environment are important. Differently from the Dolev-Yao intruder, a timed intruder needs to wait for the message to arrive before he can learn it. [39] proved an upper-bound on the number of timed intruders. Our tool implements this strategy. However, for the examples considered here, a single Dolev-Yao intruder is enough.

# 5 Timed Trace Equivalence

Our goal now is to determine when two configurations:

$$C_I = \langle P_1, IK_1, DC_1, EQ_1, TC_1 \rangle @ tG \quad \text{and} \quad C'_I = \langle P'_1, IK'_1, DC'_1, EQ'_1, TC'_1 \rangle @ tG'$$

cannot be distinguished by the Dolev-Yao intruder. That is, for any trace starting from $C_I$ there is an equivalent trace starting from $C'_I$. Intuitively, the intruder participates in the same interactions (sends and receives) with the same timing. The following definition specifies observables which collect the necessary information from a trace:

**Definition 9.** Let $\tau = C_1 \xrightarrow{h_1} C_2 \xrightarrow{h_2} \cdots \xrightarrow{h_n} C_n$ be a timed trace. Its observable is the tuple $\langle tI, L, IK_n, DC_n, EQ_n, TC_n \rangle$, where $tI$ is the global time at configuration $C_1$, $L$ is the sequence of non-empty labels in $\tau$. Let $C$ be a configuration. Let $T(C)$ be the set of all traces with initial configuration $C$. The observables of $C$ is $O(C) = \{ O_\tau | \tau \in T(C) \}$, that is, the set of all observables of traces starting from $C$.

Two configurations are trace equivalent if their observables are equivalent.

**Definition 10.** A configuration $C$ approximates a configuration $C'$, written $C \preceq C'$ if for any $O \in O(C)$ there exists an equivalent observable $O' \in O(C')$, that is, $O \sim O'$ (Definition 11). The configurations are observationally equivalent, written $C \sim C'$, if and only if $C \preceq C'$ and $C' \preceq C$.

**Definition 11.** Consider the observables $O = \langle tI, L, IK, DC, EQ, TC \rangle @ tG$ and $O' = \langle tI', L', IK', DC', EQ', TC' \rangle @ tG'$, such that

$$L = \langle (\pm_1 ms_1 @ tG_1) \ldots (\pm_p ms_p @ tG_p) \rangle \quad \text{and} \quad L' = \langle (\pm'_1 ms'_1 @ tG'_1) \ldots (\pm'_n ms'_n @ tG'_n) \rangle$$

The observation $O$ is equivalent to $O'$, written $O \sim O'$ if the following conditions are all true:

1. $p = n = N$, that is, they have the same length $N$;
2. \( \pm_i = \pm'_i \) for all \( 1 \leq i \leq N \), that is, have the same label type;
3. The messages observed are equivalent, that is, \( \langle ms_1, \ldots, ms_N \rangle \sim_{O,C} \langle ms'_1, \ldots, ms'_N \rangle \);
4. Assume \( \bar{t} \) and \( \bar{w} \) are the set of time symbols in \( TC \) and \( TC' \), respectively. These sets of time symbols are assumed disjoint without loss of generality. The following formulas are tautologies:

\[
\forall \bar{t}. \left[ TC \Rightarrow \exists \bar{t}' \left[ TC' \land tG_1 = tG'_1 \land \cdots \land tG_N = tG'_N \right] \right]
\]

\[
\forall \bar{t}' . \left[ TC' \Rightarrow \exists \bar{t}. \left[ TC \land tG_1 = tG'_1 \land \cdots \land tG_N = tG'_N \right] \right]
\]

The first two conditions are clear. If two observables differ on the number of observations or they differ on their types, then they can be distinguished. The third condition specifies that the terms observed shall be equivalent. Here, we do not specify its definition, as any definition which considers the Dolev-Yao intruder capabilities in the literature could be, in principle, used. Our technical report describes one such possible definition. Notice, however, that the knowledge of the intruder at the end of the traces shall be used, as he may have learned keys allowing him to distinguish more terms.

The fourth condition involves the timing aspects of the protocols. Intuitively, two observables are equivalent, that is, not distinguishable by the intruder, whenever the timings of when messages are observed are identical. Take the first clause below:

\[
\forall \bar{t}. \left[ TC \Rightarrow \exists \bar{t}' \left[ TC' \land tG_1 = tG'_1 \land \cdots \land tG_N = tG'_N \right] \right]
\]

It specifies that for all instances of \( \bar{t} \) that satisfy the constraints \( TC \) in \( O \), it is possible to find instances of \( \bar{t}' \) that satisfy the constraints \( TC' \) in \( O' \), that is, are valid instances, and moreover, the times of the observed messages are identical.

**Example**  Consider the passport protocol role, \( P \), described in Example 5. Moreover, consider the following two initial configurations:

\[
C_I = \langle \{ 0 \mid P \mid \{ k_M, k_E \} \}, \{ c_{enc}, c_{mac} \}, \emptyset, \emptyset, tG = 0 \rangle @ tG
\]

\[
C_I' = \langle \{ 0 \mid P \mid \{ k_M, k_E \} \}, \emptyset, \emptyset, tG' = 0 \rangle @ tG'
\]

where in \( C_I \) the intruder has already eavesdropped a communication between the passport and the identification machine, learning the constants, \( c_{enc} = e(n_{old}, k_E) \) and \( c_{mac} = e(c_{enc}, k_M) \). Notice that these messages contain a nonce generated during the first encounter of the intruder with the passport.

This means that there is a trace starting from \( C_I \) where after the passport generating a nonce, \( n_{new} \), and sending it, the intruder responds with the pair \( \{ c_{enc}, c_{mac} \} \). At this point, the passport checks that \( c_{mac} \) is encrypted with the correct key \( k_M \), but then it checks that \( c_{enc} \) has the wrong nonce, returning the error message. Thus, the observable corresponding to this trace has the following form:

\[
L = \langle (+n_{new} @ tG_1), (-c_{sym_1}, c_{sym_2}) @ tG_2), (+error @ tG_0), \rangle
\]

\[
1K = [c_{enc}, c_{mac}, n_{new}], \quad DC = [dc(c_{sym_1}, 1K), dc(c_{sym_2}, 1K)],
\]

\[
E = [eq(c_{sym_2}, n_{sym_1}, k_M), neq(c_{sym_1}, e(n_{new}, k_E))]
\]

\[
TC = \{ tG_0 = 0, t_0 = tG_2, t_1 = t_0 + r_{mac}, t_2 = t_1 + r_{enc}, tG_6 = t_2 \} \cup \{ tG_{i+1} \geq tG_i \mid 0 \leq i \leq 5 \}
\]
From the time constraints $\mathcal{T}C$, we deduce that $tG_0 = r_{mac} + r_{enc} + tG_2$.

On the other hand, there is only one trace from $C'_f$, which yields the observable:

\[
\begin{align*}
L' &= \langle n_{enc} @ tG'_1, (-\langle \text{sym}_1, \text{sym}_2 \rangle @ tG'_2, (+\text{error} @ tG'_3) \rangle, \\
I\mathcal{K}' &= \{ n_{enc} \}, \\
\mathcal{D}C &= \{ \text{dc}(\text{sym}'_1, I\mathcal{K}''), \text{dc}(\text{sym}'_2, I\mathcal{K}'') \}, \\
\mathcal{EQ}' &= \{ \text{neq}(\text{sym}'_1, e(\text{sym}_1, K_M)) \}
\end{align*}
\]

$\mathcal{T}C' = \{ tG_0 = 0, tI'_0 = tG'_2, tI'_i = tG'_i | 0 \leq i \leq 2 \}$

From the time constraints $\mathcal{T}C'$, we can infer that $tG'_i = tG'_2 + r_{mac}$.

Clearly, the condition on the time variables for observational equivalence cannot be satisfied, as $tG_2 = tG'_2$ and $tG_0 = tG'_3$ cannot be both satisfied.

The condition Definition 11.4 assumes a powerful intruder that can distinguish observables associated with different times, even with infinitesimal differences. This may result in false positives, as our definition would flag attacks that in practice are not possible to carry out. We can, however, relax the condition Definition 11.4 and consider observables equivalent even if the time of observables are different within some range. We replace the condition Definition 11.4 by the following:

\[
\forall tt. [\mathcal{T}C \Rightarrow \exists tt'. [\mathcal{T}C' \land |tG_1 - tG'_1| \leq \epsilon \land \cdots \land |tG_N - tG'_N| \leq \epsilon]]
\]

\[
\forall tt'. [\mathcal{T}C' \Rightarrow \exists tt. [\mathcal{T}C \land |tG_1 - tG'_1| \leq \epsilon \land \cdots \land |tG_N - tG'_N| \leq \epsilon]]
\]

The greater the value of $\epsilon$, the weaker the capability of the intruder to measure the timing of observables. For example, if $\epsilon$ is the time for encrypting terms, the corresponding weak intruder could not carry out the passport attack.

### 5.1 Automating the Check of Time Approximation

For Condition 11.4, we reduce the formulas to formulas for which existing solvers can be used [18], namely formulas of the form $\exists V$:

\[
\forall tt. [\mathcal{T}C \Rightarrow \exists tt'. [\mathcal{T}C' \land tG_1 = tG'_1 \land \cdots \land tG_N = tG'_N]]
\]

is a tautology

\[
\Leftrightarrow \neg \forall tt. [\mathcal{T}C \Rightarrow \exists tt'. [\mathcal{T}C' \land tG_1 = tG'_1 \land \cdots \land tG_N = tG'_N]]
\]

is unsat

\[
\Leftrightarrow \exists tt. [\mathcal{T}C \land \forall tt'. [\mathcal{T}C' \Rightarrow \neg [tG_1 = tG'_1 \land \cdots \land tG_N = tG'_N]]]
\]

is unsat

### 6 Experimental Results

We implemented a tool that checks for timed observational equivalence. Its architecture is depicted in Figure 2. It is constructed using Maude [12] and the Yices SMT solver [17], coordinated by the IOP framework [31]. In particular, we use Yices for solving $\exists V$ formulas where all time variables have type Real (which is decidable [18]).

- **Maude**: We implemented in Maude all the machinery necessary for specifying timed protocols as well as checking the term equivalence of observables. Since Alpha version 111 Maude provides a builtin function to call an SMT solver to check satisfiability of constraints supported by the SMT API. This allows the implementation of

---

8 Initially this was implemented using CVC4 [4]. Since Alpha 114 there is also the option to use Yices2.
Rewriting Modulo SMT by using conditional rewrite rules that are only allowed to rewrite if the resulting constraint set is satisfiable. Our rewrite rules include as a condition a call to the SMT solver to check satisfiability of the time constraints whenever firing the rule would add to the constraints.

– **YICES-EF**: Since the SMT standard interface does not provide an API for checking \( \exists \forall \) formulas needed for proving time equivalence, we integrated our Maude machinery with YICES-EF, a wrapper for Yices2 that translates \( \exists \forall \) formulas into the Yices2 language and calls Yices to check the satisfiability of such formulas.

Communication between Maude and YICES-EF is implemented using the IOP message passing framework [31]. Given two initial configuration \( C \) and \( C' \) to be checked for their timed trace equivalence, the user uses IOP to send a command to the Maude+Yices tool to enumerate all observables for \( C \) and \( C' \), then compute for each observable \( O \) of \( C \) the set of term-equivalent observables \( \{O_1', \ldots, O_n'\} \) of \( C' \) and vice-versa. If Maude finds some observable of \( C \) that does not have at least one match, that is, \( n = 0 \), \( C \) and \( C' \) are not equivalent. (Similarly for matching some \( C' \).) Otherwise, for each \( O, O' \) match Maude asks YICES-EF to check the timing equivalence condition. If for each \( O \) there is at least one \( O' \) such that the timing equivalence condition is satisfied (YICES-EF returns \( \text{Sat} \)), then \( C \) approximates \( C' \). In the same way, Maude also checks whether \( C' \) approximates \( C \). If both directions succeed then the two configurations are timed trace equivalent. If either direction fails then they are not equivalent. To make the above request to YICES-EF, Maude builds (a representation of) the formula described in Section 5.1 for checking for the timing equivalence of \( O \) with \( O'_i \), which is transformed by YICES-EF to Yices2 input format.

**Experimental Results** We carried out the following experiments:

– **Red Pill Example**: Consider the timed protocol role specified in Example 6. We checked whether it is possible for an intruder to distinguish whether an application is running over a virtual machine or not. That is, we checked whether an initial configuration with a player running an application over a virtual machine is timed equivalent to the initial configuration with a player running the same application over a non-virtual machine.
Table 1 summarizes the results of our experiments. Our tool was able to (correctly) identify the cases when the given configurations are timed observational equivalent. More impressive, however, is the number of states and observables it needed to traverse for doing so. In all experiments the number of states in the whole search tree was less than 140 states and the number of observables were less than 40. This is a very small number when compared to usual applications in Maude (which can handle thousands of states even when using Rewriting Modulo SMT [39]). This demonstrates the advantage of representing timing symbolically. As expected the number of observables for the passport example were greater as its configurations had two protocol sessions, while in the remaining experiments configurations have only one protocol session. Finally, since the number of observables was small, the number of calls to Yices was small and therefore, verification for all experiments took less than a few seconds.

7 Related and Future Work

In this paper we introduce a novel definition of timed trace equivalence for security protocols using symbolic time constraints. We demonstrate how symbolic time equivalence can be proved automatically with the use of Rewriting Modulo SMT and existing SMT-solvers assuming a bounded number of protocol sessions. The combination of such constraints with Rewriting Modulo SMT greatly reduces the number of states required
to enumerate all traces. We implemented the machinery for proving the timed observational equivalence and showed experimentally with some proof-of-concept examples that our technique is practical.

For future work, we will be integrating the machinery developed here with the Maude’s generic unification capability [16] which can be used for the analysis of security protocols that use a wide range of crypto algebras and weaker notions of encryption. This will take advantage of the independence of the symbolic message constraints and the time constraints.

In our current approach, we verify scenarios with a bounded number of protocol sessions. We are investigating how to adapt techniques, such as Narrowing used in Maude-NPA [20], to support time constraints.

We are also investigating notions of timed observational equivalence and their relation to the timed trace equivalence proposed here. It seems possible to define timed observational equivalence by using the intruder upper bound result in our previous work [39] with the notion of trace equivalence towards the definition of a notion of timed observational equivalence, i.e., with quantification over intruder contexts, that can be solved using SMT-solvers.

Furthermore, we plan to investigate how to use SMT solvers in order to answer questions such as, what is the weakest intruder that can carry out an attack. In particular, given the more relaxed notion of time equivalence described at the end of Section 5, we can consider $\epsilon$ as a variable to be maximized by an SMT optimizer.

Related Work: The literature on symbolic verification is vast [5,9,14,20,22]. However, most of this work uses symbolic reasoning for proving either reachability properties or properties not involving timing aspects.

One exception is the work of [14]. Indeed, for the observational equivalence involving terms, we have been heavily inspired by [14], but there are some differences. The main difference is that our timed protocols includes both message and time symbols. We also implemented our machinery for term equivalence in Maude and use SMT-solvers for search (Rewriting Modulo SMT) and proving the timing equivalence.

Cheval and Cortier [10] propose a definition of timed equivalence reducing it to other notions of equivalence taking into account the length of messages. We take a different approach by using timed constraints and SMT-Solvers. This allows us to relate time symbols using inequalities, e.g., $t_1 \geq t_2 + 10$, which can be solved by off-the-shelf SMT solvers. Moreover, time constraints can also specify timing aspects not directly related to the length of messages, such as, the properties used in Distance Bounding protocols. Finally, as we illustrate one can also define coarser definitions of time equivalence, involving intruders with weaker time measuring mechanisms, potentially leading to less false positives.

Gazeau et al. [22] demonstrate how to automate the proof of observational equivalence of protocols that may contain branching and xor. While we allow for branching, we do not yet consider theories involving xor. However, we do consider timing aspects, which is not considered in [22]. Thus these works are complementary. As described above, we expect in the future to support xor (and other equational theories) by using the built-in Maude matching and unification functionality [16].
Finally, there have been other frameworks for the verification of timing properties of systems [6, 21, 25, 28, 30]. A main difference is that the properties verified were reachability properties and not equivalence notions involving timing aspects.

Acknowledgments. We thank the anonymous reviewer for careful reading and helpful suggestions for improvement. Nigam was partially supported by NRL grant N0017317-1-G002 and by CNPq grant 303909/2018-8. Talcott was partly supported by ONR grant N00014-15-1-2202 and NRL grant N0017317-1-G002.

References