Symbolic Timed Trace Equivalence

Vivek Nigam[1,2], Carolyn Talcott[3] and Abraão Aires Urquiza[1]

[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 collegue 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.*, $\mathtt{tt}_1 \geq \mathtt{tt}_2 + 10$;
– **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 machines 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, $\mathcal{T}$, player names, $\mathcal{P}$, nonces, $\mathcal{N}$, symmetric keys, $\mathcal{K}$, symbols, Syms, and variables, $\mathcal{V}$, where $\mathcal{T}$, $\mathcal{P}$, $\mathcal{N}$, $\mathcal{K}$, Syms and $\mathcal{V}$ are disjoint. Below v_p repre-

sents a variable of sort player.

Basic Constants:

$\mathsf{c} :=$	$\mathsf{t} \in \mathcal{T}$	Text Constants
	$\mid \mathsf{p} \in \mathcal{P}$	Player Names
	$\mid \mathsf{n} \in \mathcal{N}$	Nonces

Symbols:

$\mathsf{sym} :=$	$\mid \mathsf{sym} \in \mathsf{Syms}$	Symbol

Keys:

$\mathsf{k} :=$	$\mathsf{symk} \in \mathcal{K}$	Symmetric keys
	$\mid \mathsf{pk(p)} \mid \mathsf{pk}(\mathsf{v}_p)$	Public key of a player
	$\mid \mathsf{sk(p)} \mid \mathsf{sk}(\mathsf{v}_p)$	Secret key of a player

Terms:

$\mathsf{m} :=$	c	Basic constants
	$\mid \mathsf{k}$	Keys
	$\mid \mathsf{v} \in \mathcal{V}$	Variables
	$\mid \mathsf{sym} \in \mathsf{Syms}$	Symbols
	$\mid \mathsf{e(m, k)}$	Encrypted term
	$\mid \langle \mathsf{m}_1, \ldots, \mathsf{m}_n \rangle$	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. $\mathsf{ms}, \mathsf{ms}_1, \mathsf{ms}_2, \ldots$ will range over symbolic terms.

Time Assume a time signature, Ξ, containing a set of numbers, $\mathsf{r}_1, \mathsf{r}_2, \ldots$, a set of time variables, $\mathsf{tt}_1, \mathsf{tt}_2, \ldots$, including the special variable cur, and a set of pre-defined function symbols, including, $+, -, \times, /, \mathsf{floor}, \mathsf{ceiling}$.

Time Expressions are constructed inductively from numbers and variables by applying function symbols to time expressions. For example $\mathsf{ceiling}((2 + \mathsf{tt} + \mathsf{cur})/10)$ is a Time Expression. The symbols $\mathsf{tr}_1, \mathsf{tr}_2, \ldots$ range over Time Expressions. The time variable 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 Ξ, 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 Ξ be a time signature. The set of symbolic time constraints is constructed using time expressions as follows: Let $\mathsf{tr}_1, \mathsf{tr}_2$ be time expressions, then*

$$\mathsf{tr}_1 = \mathsf{tr}_2, \quad \mathsf{tr}_1 \geq \mathsf{tr}_2 \quad \mathsf{tr}_1 > \mathsf{tr}_2, \quad \mathsf{tr}_1 < \mathsf{tr}_2, \text{ and } \mathsf{tr}_1 \leq \mathsf{tr}_2$$

are symbolic time constraints.

For example, $\mathsf{cur} + 10 < \mathsf{floor}(\mathsf{tt} - 5)$ is a time constraint. $\mathsf{tc}, \mathsf{tc}_1, \mathsf{tc}_2, \ldots$ will range over time constraints.

Intuitively, given a set of time constraints $\mathcal{TC}$, 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 $\{\mathsf{tt}_1 \leq 2, \mathsf{tt}_2 \geq 1 + \mathsf{tt}_1\}$ has an infinite number of models, *e.g.*, $[\mathsf{tt}_1 \mapsto 1.9, \mathsf{tt}_2 \mapsto 3.1415]$.

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,* $\mathsf{pl} \in \mathcal{TL}$*, is generated by the following grammar:*

nil	*Empty Protocol*
$\mid (\mathsf{new\ v\ \#\ tc}), \mathsf{pl}$	*Fresh Constant*
$\mid (+\mathsf{m\ \#\ tc}), \mathsf{pl}$	*Timed Message Send*
$\mid (-\mathsf{m\ \#\ tc}), \mathsf{pl}$	*Timed Message Receive*
$\mid (\mathsf{if}\ (\mathsf{m}_1 := \mathsf{m}_2)\ \#\ \mathsf{tc}\ \mathsf{then}\ \mathsf{pl}_1\ \mathsf{else}\ \mathsf{pl}_2)$	*Timed Conditional*

Intuitively, new generates a fresh value binding it to the variable v, $(+\mathsf{m}\ \#\ \mathsf{tc})$ denotes sending the term m and $(-\mathsf{m}\ \#\ \mathsf{tc})$ denotes receiving a term matching m. For the term $(\mathsf{if}\ \mathsf{m}_1 := \mathsf{m}_2\ \#\ \mathsf{tc}\ \mathsf{then}\ \mathsf{pl}_1\ \mathsf{else}\ \mathsf{pl}_2)$, 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 pl_1 and otherwise to execute pl_2. Moreover, the variables in m_2 are instantiated with the witnessing matching substitution in pl_1. We also assume that pl_2 does not contain variables in ms_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:

$$Alice(Z) := (\mathsf{new}\ N_a), (+\mathsf{e}(\langle N_a, alice\rangle, \mathsf{pk}(Z))), (-\mathsf{e}(\langle N_a, Y\rangle, \mathsf{pk}(alice)\})), (+\mathsf{e}(Y, \mathsf{pk}(Z)))$$
$$Bob := (-\mathsf{e}(\langle X, Z\rangle, \mathsf{pk}(bob))), (\mathsf{new}\ N_b), (+\mathsf{e}(\langle X, N_b\rangle, \mathsf{pk}(Z)\})), (-\mathsf{e}(N_b, \mathsf{pk}(bob)))$$

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):

$$Alice(Z) := (\mathsf{new}\ N_a), (+\mathsf{e}(\langle N_a, alice\rangle, \mathsf{pk}(Z))), (-\mathsf{v}),$$
$$\mathsf{if}\ \mathsf{v} := \mathsf{e}(\langle N_a, Y\rangle, \mathsf{pk}(alice)\})\ \mathsf{then}\ (+\mathsf{e}(Y, \mathsf{pk}(Z)))\ \mathsf{else}\ (+error)$$

Here, Alice checks whether the received message v 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:

$$(\mathsf{new}\ \mathsf{v}), (+\mathsf{v}\ \#\ \mathsf{tt} = \mathsf{cur}), (-\mathsf{v}\ \#\ \mathsf{cur} \leq \mathsf{tt} + 4)$$

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:

$$(\mathsf{new}\ \mathsf{v}), (+\mathsf{v}\ \#\ \mathsf{tt} = \mathsf{cur}), (-\mathsf{v}_2\ \#\ \mathsf{tt}' = \mathsf{cur}),$$
$$(\mathsf{if}\ (\mathsf{v} := \mathsf{v}_2\ \#\ \mathsf{tt} + 4 \geq \mathsf{tt}'))\ \mathsf{then}\ (+\mathit{ok}\ \#\ \mathsf{cur} < \mathsf{tt}' + 2)\ \mathsf{else}\ (+\mathit{ko}\ \#\ \mathsf{cur} < \mathsf{tt}' + 2)$$

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.

$$(\mathsf{new}\ \mathsf{v}), (+\mathsf{v}), (-\langle \mathsf{v}_{enc}, \mathsf{v}_{mac} \rangle\ \#\ \mathsf{tt}_0 = \mathsf{cur})$$
$$\mathsf{if}\ (\mathsf{v}_{mac} := \mathsf{e}(\mathsf{v}_{enc}, \mathsf{k}_M))\ \#\ \mathsf{tt}_1 = \mathsf{tt}_0 + \mathsf{r}_{mac}\ \mathsf{then}$$
$$\quad \mathsf{if}\ (\mathsf{v}_{enc} := \mathsf{e}(\mathsf{v}, \mathsf{k}_E))\ \#\ \mathsf{tt}_2 = \mathsf{tt}_1 + \mathsf{r}_{enc})\ \mathsf{then}\ (+\mathit{done}\ \#\ \mathsf{cur} = \mathsf{tt}_2)\ \mathsf{else}\ (+\mathit{error}\ \#\ \mathsf{cur} = \mathsf{tt}_2)$$
$$\mathsf{else}\ (+\mathit{error}\ \#\ \mathsf{cur} = \mathsf{tt}_1)$$

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 tt_0. It then checks whether the first component v_{mac} is of the form $\mathsf{e}(\mathsf{v}_{enc}, \mathsf{k}_M)$, *i.e.*, it is the correct MAC. This operation takes r_{mac} time units. The time variable tt_1 is equal to the time $\mathsf{tt}_0 + \mathsf{r}_{mac}$, *i.e.*, the time when the message was received plus the MAC check duration. If the MAC is not correct, an *error* message is sent at time 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 $\mathsf{e}(\mathsf{v}, \mathsf{k}_E)$, which takes (a longer) time r_{enc}. If so it sends the *done* message, otherwise the *error* message, both at time tt_2 which is $\mathsf{tt}_1 + \mathsf{r}_{enc}$.

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 $\mathsf{tt}_0 + \mathsf{r}_{mac-} \leq \mathsf{tt}_1 \leq \mathsf{tt}_0 + \mathsf{r}_{mac+}$ express that it takes a time between r_{mac-} and r_{mac+} to check the 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 dBase to complete the exchange of the baseline messages.

$$(-(\mathsf{baseline\_req})\ \#\ \mathsf{tt}_0 = \mathsf{cur}), (+(\mathsf{baseline\_done})\ \#\ \mathsf{cur} = \mathsf{tt}_0 + \mathsf{dBase}),$$
$$(-(\mathsf{diff\_req})\ \#\ \mathsf{tt}_1 = \mathsf{cur}), (+(\mathsf{diff\_done})\ \#\ \mathsf{cur} = \mathsf{tt}_1 + \mathsf{dAppl})$$

Then the part of the protocol that depends on the application starts. We abstract this part using the messages diff_req and diff_done. If the application is running over a virtual machine, then dAppl takes dVirtual time units; otherwise dAppl takes dReal time units, where `dVirtual > dReal`.

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 diff_req and diff_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 KB. 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_ν, only known to the agent.

$$-(\mathsf{v}), \mathsf{if}\ \mathsf{v} := \langle \mathsf{hello}, \mathsf{e}(\{\mathsf{hello}, \mathsf{v}_n, \mathsf{k}_A\}, \mathsf{k}_G)\rangle\ \#\ \mathsf{tt}_1 = \mathsf{cur} + \mathsf{dEnc}\ \mathsf{then}$$
$$\mathsf{if}\ (\mathsf{k}_G := KB\ \#\ \mathsf{tt}_2 = \mathsf{tt}_1 + \mathsf{dChk})\ \mathsf{then}\ + (\langle \mathsf{ack}, \mathsf{e}(\mathsf{rsp}, \mathsf{k}_A)\rangle)\ \#\ \mathsf{cur} = \mathsf{tt}_1 + \mathsf{dCrt}$$
$$\quad \mathsf{else}\ (\mathsf{new}\ \mathsf{k}_\nu), +(\langle \mathsf{ack}, \mathsf{e}(\mathsf{decoy}, \mathsf{k}_\nu)\rangle)\ \#\ \mathsf{cur} = \mathsf{tt}_1$$
$$\mathsf{else}\ (\mathsf{new}\ \mathsf{k}_\nu), +(\langle \mathsf{ack}, \mathsf{e}(\mathsf{decoy}, \mathsf{k}_\nu)\rangle)\ \#\ \mathsf{cur} = \mathsf{tt}_1$$

Here dEnc, dChk and dCrt 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].[5]

4.1 Symbolic Term Constraints

We will use two types of (capture avoiding) substitutions. *Variable substitutions* written $\mathsf{sb}, \mathsf{sb}_1, \mathsf{sb}_2, \ldots$ which are maps from variables to symbolic terms $\mathsf{sb} = [\mathsf{v}_1 \mapsto \mathsf{ms}_1, \mathsf{v}_2 \mapsto \mathsf{ms}_2, \ldots, \mathsf{v}_n \mapsto \mathsf{ms}_n]$. *Symbol substitutions* written $\mathsf{ssb}, \mathsf{ssb}_1, \mathsf{ssb}_2, \ldots$ mapping symbols to symbolic terms $\mathsf{ssb} = [\mathsf{sym}_1 \mapsto \mathsf{ms}_1, \ldots, \mathsf{sym}_n \mapsto \mathsf{ms}_n]$.

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

Definition 3. *A derivability constraint has the form* $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$*, where* $\mathcal{S}$ *is a set of symbolic terms and* sym *a symbol. This constraint denotes that* sym *can be any (symbolic) term derived from* $\mathcal{S}$*.* [6]

For example, the derivability constraint

$$\mathsf{dc}(\mathsf{sym}, \{\mathit{alice}, \mathsf{n}_1, \mathsf{e}(\mathsf{sym}_2, \mathsf{sk}(\mathit{alice})), \mathsf{pk}(\mathit{bob})\})$$

specifies that sym may be instantiated by, *e.g.*, the terms $\langle \mathit{alice}, \mathsf{e}(\mathsf{sym}_2, \mathsf{sk}(\mathit{alice}))\rangle$, $\langle \mathit{alice}, \mathsf{n}_1\rangle$, $\mathsf{e}(\mathit{alice}, \mathsf{pk}(\mathit{bob}))$, $\mathsf{e}(\langle \mathit{alice}, \mathit{bob}\rangle, \mathsf{pk}(\mathit{bob}))$ and so on.

Notice that any $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$ denotes a infinite number of symbolic terms due to the tupling closure. We will abuse notation and use $\mathsf{ms} \in \mathsf{dc}(\mathsf{sym}, \mathcal{S})$ to denote that the symbolic term ms is in the set of terms that can be derived from $\mathcal{S}$. Moreover, we assume that for any given set of derivability constraints $\mathcal{DC}$, 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 $\mathsf{ms} \in \mathcal{DC}(\mathsf{ms}')$. Roughly, this means that ms can be obtained by repeated substitution of a symbol, sym in ms' by one of its instantiations according to $\mathsf{dc}(\mathsf{sym}, \mathcal{S}) \in \mathcal{DC}$. For example, assume that $\mathcal{DC}$ contains $\mathsf{dc}(\mathsf{sym}_1, \{\mathsf{t}_1\})$, $\mathsf{dc}(\mathsf{sym}_2, \{\mathsf{t}_2\})$, then $\langle \mathsf{t}_1, \mathsf{t}_2\rangle \in \mathcal{DC}(\langle \mathsf{sym}_1, \mathsf{sym}_2\rangle)$. The technical report [38] describe an algorithm to decide $\mathsf{ms} \in \mathcal{DC}(\mathsf{ms}')$[7].

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

The following definition specifies the second type of symbolic term constraints called comparison constraints.

[5] The accompanying implementation can be found at https://github.com/SRI-CSL/VCPublic/obseq.git.

[6] $\mathcal{S}$ always includes guessables–names, text, fresh nonces, ... Guessables are left implicit in our examples.

[7] Strictly, $\mathcal{DC}$ needs to satisfy some conditions in order for this membership relation to be well-defined. For example, the symbol dependency graph of $\mathcal{DC}$ shall be acyclic. We assume that this relation is undefined whenever this is not the case.

Definition 5. *A comparison constraint is either an equality constraint* $\mathsf{eq}(\mathsf{ms}_1, \mathsf{ms}_2)$ *or an inequality constraint* $\mathsf{neq}(\mathsf{ms}_1, \mathsf{ms}_2)$*, where* $\mathsf{ms}_1, \mathsf{ms}_2$ *are symbolic terms.*

A set $\mathcal{EQ}$ of comparison constraints is interpreted as a conjunction of constraints. We write $\mathcal{DC} \vDash \mathcal{EQ}$ 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 $\mathsf{dc}(\mathsf{sym}_1, \{\mathsf{t}_1\}), \mathsf{dc}(\mathsf{sym}_2, \{\mathsf{t}_2\}) \in \mathcal{DC}$ and $\mathcal{EQ} = \{\mathsf{eq}(\mathsf{sym}_1, \mathsf{sym}_2)\}$, the symbolic term $\langle \mathsf{sym}_1, \mathsf{sym}_2 \rangle$ may represent $\langle \mathsf{t}_1, \mathsf{t}_2 \rangle$ when considering only $\mathcal{DC}$, but not when considering both $\mathcal{DC}$ and $\mathcal{EQ}$, as it falsifies the constraint that sym_1 and sym_2 are equal, witnessed by the matching substitution $\theta = \{\mathsf{sym}_1 \mapsto \mathsf{t}_1, \mathsf{sym}_2 \mapsto \mathsf{t}_2\}$. We write $\mathsf{ms} \in \mathcal{DC}(\mathsf{ms}') \mid_{\mathcal{EQ}}$ to denote when the symbolic term ms' can represent the symbolic term ms assuming the constraints $\mathcal{DC}$ and $\mathcal{EQ}$. Algorithms to decide $\mathcal{DC} \vDash \mathcal{EQ}$ and $\mathsf{ms} \in \mathcal{DC}(\mathsf{ms}') \mid_{\mathcal{EQ}}$ 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* $\mathcal{IK}$ *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 $\mathcal{IK}$.

For example, consider the term $\mathsf{m} = \mathsf{e}(\langle \mathsf{v}_1, \mathsf{sym}, \mathsf{v}_1, \mathsf{v}_2 \rangle, \mathsf{k})$ (which is expected as input by an honest player). Here v_1 and v_2 are variables and sym is constrained by derivability constraints $\mathcal{DC}$. We create two fresh symbols sym_1 and sym_2 for, respectively, the variables v_1 and v_2. Letting $\mathsf{sb} = [\mathsf{v}_1 \mapsto \mathsf{sym}_1, \mathsf{v}_2 \mapsto \mathsf{sym}_2]$, we obtain $\mathsf{ms} = \mathsf{sb}[\mathsf{m}] = \mathsf{e}(\langle \mathsf{sym}_1, \mathsf{sym}, \mathsf{sym}_1, \mathsf{sym}_2 \rangle, \mathsf{k})$.

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 $\mathcal{IK}$.

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

$$\{\mathsf{sb}, \{\mathsf{ssb}_1, \mathcal{DC}_1\} \ldots \{\mathsf{ssb}_k, \mathcal{DC}_k\}\}$$

where sb maps the variables of m to fresh symbols, and each $\{\mathsf{ssb}_i, \mathcal{DC}_i\}$ is a solution to the problem above for $\mathsf{ms} = \mathsf{sb}[\mathsf{m}]$. 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 $\mathcal{IK}$ or deriving m from terms in $\mathcal{IK}$. The following example illustrates the different cases involved:

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

– Case 1 (matching with a term in $\mathcal{IK}$): Assume:

$$\mathcal{IK} = \{\mathsf{e}(\langle \mathsf{n}_a, \mathsf{sym}_1 \rangle, \mathsf{k})\} \qquad \mathcal{DC} = \mathsf{dc}(\mathsf{sym}, \{\mathsf{n}_a, \mathsf{n}_c\})\ \mathsf{dc}(\mathsf{sym}_1, \mathcal{S})$$

Then the solution of sgen is:

$$\{\mathsf{sb}, \{[\mathsf{sym}_\mathsf{v} \mapsto \mathsf{n}_a, \mathsf{sym} \mapsto \mathsf{sym}_1], \mathsf{dc}(\mathsf{sym}_1, \{\mathsf{n}_a, \mathsf{n}_c\} \cap \mathcal{S})\}\}$$

where $\mathsf{sb} = [\mathsf{v} \mapsto \mathsf{sym}_\mathsf{v}]$ and sym_v is a fresh symbol. Notice that since sym_v is mapped to a particular term (n_a), 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 $\mathsf{dc}(\mathsf{sym}, \mathcal{S})$;

– Case 2 (constructing terms from $\mathcal{IK}$): Assume that $\mathsf{k} \in \mathcal{IK}$ and $\mathcal{IK}$ has no encryption terms. Then the solution of sgen is:

$$\{[\mathsf{v} \mapsto \mathsf{sym}_\mathsf{v}], \{[], \mathcal{DC} \cup \{\mathsf{dc}(\mathsf{sym}_\mathsf{v}, \mathcal{IK})\}\}\}$$

which corresponds to generating the term $\mathsf{e}(\{\mathsf{sym}_v, \mathsf{sym}\}, \mathsf{k})$. Moreover, sym_v can be any term derivable from the intruder's knowledge.

– Case 3 (No Solution): Assume that $\mathcal{IK} = \{\mathsf{e}(\langle \mathsf{n}_a, \mathsf{n}_b \rangle, \mathsf{k})\}$ and $\mathcal{DC} = \mathsf{dc}(\mathsf{sym}, \{\mathsf{n}_a, \mathsf{n}_c\})$. Since sym cannot be instantiated to n_b, the intruder cannot use the term $\mathsf{e}(\langle \mathsf{n}_a, \mathsf{n}_b \rangle, \mathsf{k})$.

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* $\langle \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}$*, where*

– $\mathcal{P}$ *is a finite set of player roles of the form* $[n \mid \mathsf{pl} \mid \mathsf{keys}]$ *composed of an identifier, n, a protocol role* pl*, and a set keys,* keys*, known to the player;*
– $\mathcal{IK}$ *is the intruder knowledge;*
– $\mathcal{DC}$ *is a set of derivability constraints;*
– $\mathcal{EQ}$ *is a set of comparison constraints;*
– $\mathcal{TC}$ *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^{ν}. 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 $\mathsf{sgen}(\mathsf{m}, \mathcal{IK}, \mathcal{DC})$ returns the variable substitution sb and a set of solutions $\{\mathsf{ssb}, \mathcal{DC}_1\}$ 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 $\mathsf{ms} = \mathsf{ssb}[\mathsf{sb}[\mathsf{m}]]$ is encrypted with keys that can be decrypted by the honest participant. This is specified by the function *isReceivable*. 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 **If-true** checks whether the terms ms_1 and ms_2 can be matched given the constraints $\mathcal{DC}$. This is done by the function sgenB which uses sgen. It first matches the structure of the terms in the matching problem $\mathsf{ms}_1 = \mathsf{ms}_2$, where symbols are considered as variables. If they cannot be matched, then the empty solution set is returned. Otherwise, there is a witnessing match $\mathsf{sym}_1 \mapsto \mathsf{ms}_1, \ldots, \mathsf{sym}_n \mapsto \mathsf{ms}_n$. It then calls sgen on the terms $\langle \mathsf{sym}_1, \ldots, \mathsf{sym}_n \rangle$ and $\langle \mathsf{ms}_1, \ldots, \mathsf{ms}_n \rangle$ returning its output. The rule **If-true** then adds the equality constraint to the set of comparison constraints.

Finally, the rule **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), $\mathcal{IK}_0 = \{\mathsf{sk}(\mathit{eve})\}$ and there are no symbols $\mathcal{DC} = \emptyset$. An execution of Alice's protocol role is as follows. Alice creates a fresh constant N_a and sends the message $\mathsf{e}(\langle N_a, \mathit{alice} \rangle, \mathsf{pk}(\mathit{eve}))$. At this point, the intruder knowledge is:

$$\mathcal{IK}_1 = \mathcal{IK}_0 \cup \{N_a\}$$

He now can send a message to *Bob*, namely $\mathsf{e}(\langle \mathsf{sym}_1, \mathsf{sym}_2 \rangle, \mathsf{pk}(\mathit{bob}))$ where $\mathsf{sym}_1, \mathsf{sym}_2$ are fresh and constrained $\mathcal{DC}_1 = \{\mathsf{dc}(\mathsf{sym}_1, \mathcal{IK}_1), \mathsf{dc}(\mathsf{sym}_2, \mathcal{IK}_1)\}$. At this point, Bob creates a fresh value N_b and sends the message $\mathsf{e}(\langle \mathsf{sym}_1, N_b \rangle, \mathsf{pk}(\mathsf{sym}_2)\})$. The intruder learns this message:

$$\mathcal{IK}_2 = \mathcal{IK}_1 \cup \{\mathsf{e}(\langle \mathsf{sym}_1, N_b \rangle, \mathsf{pk}(\mathsf{sym}_2))\}$$

Now, the intruder can fool alice by sending her a message of the form $\mathsf{e}(\langle N_a, Y \rangle, \mathsf{pk}(\mathit{alice})\})$. We create a fresh symbol sym_3 for Y obtaining $\mathsf{e}(\langle N_a, \mathsf{sym}_3 \rangle, \mathsf{pk}(\mathit{alice})\})$ and attempt to generate this message from $\mathcal{IK}_2$ using sgen. Indeed we can generate this message using $\mathsf{e}(\langle \mathsf{sym}_1, N_b \rangle, \mathsf{pk}(\mathsf{sym}_2)\}) \in \mathcal{IK}_2$. This generates the $\mathsf{ssb} = [\mathsf{sym}_1 \mapsto N_a, \mathsf{sym}_2 \mapsto \mathit{alice}, \mathsf{sym}_3 \mapsto N_b]$. This substitution is consistent with $\mathcal{DC}_1$. 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 $\mathcal{TC}_1$ are added to the configuration's constraint set. These time constraints are obtained by replacing cur in tc with tG_1 together with the constraint

New: $\langle [n \mid (\mathsf{new\ v\ \#\ tc}), \mathsf{pl} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC}\rangle @ \mathsf{tG}_0$
$\longrightarrow \langle [n \mid \mathsf{sb[pl]} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC}_1\rangle @ \mathsf{tG}_1$
where n^ν is a fresh nonce and $\mathsf{sb} = [\mathsf{v} \mapsto \mathsf{n}^\nu]$

Send: $\langle [n \mid (+\mathsf{ms}\ \#\ \mathsf{tc}), \mathsf{pl} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC}\rangle @ \mathsf{tG}_0$
$\longrightarrow \langle [n \mid \mathsf{pl} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK} \cup \{\mathsf{ms}\}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC}_1\rangle @ \mathsf{tG}_1$

Receive: $\langle [n \mid (-\mathsf{m}\ \#\ \mathsf{tc}), \mathsf{pl} \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC}\rangle @ \mathsf{tG}_0$
$\longrightarrow \mathsf{ssb}[\langle [n \mid \mathsf{sb[pl]} \mid \mathsf{addKeys(ms, keys)}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}_1, \mathcal{EQ}, \mathcal{TC}_1\rangle]@\mathsf{tG}_1$
where $\{\mathsf{sb}, \{\mathsf{ssb}, \mathcal{DC}_1\}\ \mathsf{css}\} := \mathsf{sgen}(\mathsf{m}, \mathcal{IK}, \mathcal{DC})$ and $\mathsf{ms} = \mathsf{ssb[sb[m]]}$ and $\mathit{isReceivable}(\mathsf{ms}, \mathsf{keys})$

If-true: $\langle [n \mid (\mathsf{if}\ (\mathsf{ms}_1 := \mathsf{ms}_2\ \#\ \mathsf{tc})\ \mathsf{then\ pl}_1\ \mathsf{else\ pl}_2) \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC}\rangle @ \mathsf{tG}_0$
$\longrightarrow \mathsf{ssb}[\langle [n \mid \mathsf{sb[pl}_1\mathsf{]} \mid \mathsf{keys}\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}_1, \mathcal{EQ} \cup \{\mathsf{eq(sb[ms}_1\mathsf{], sb[ms}_2\mathsf{])}\}, \mathcal{TC}_1\rangle]@\mathsf{tG}_1$
where $\{\mathsf{sb}, \{\mathsf{ssb}, \mathcal{DC}_1\}\ \mathsf{css}\} := \mathsf{sgenB}(\mathsf{ms}_1 = \mathsf{ms}_2, \mathcal{IK}, \mathcal{DC})$

If-false: $\langle [n \mid (\mathsf{if\ ms}_1 := \mathsf{ms}_2\ \#\ \mathsf{tc\ then\ pl}_1\ \mathsf{else\ pl}_2) \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}\rangle$
$\longrightarrow \langle [n \mid \mathsf{pl}_2 \mid \mathsf{keys}]\ \mathcal{P}, \mathcal{IK}, \mathcal{DC} \cup \mathcal{DC}', \mathcal{EQ} \cup \{\mathsf{neq(ms}_1\mathsf{, ms}_2\mathsf{)}\}\rangle$

Fig. 1. Operational semantics for basic protocols. In each rule tc_1 is the time constraint obtained by replacing cur in tc by the global time tG_1; and $\mathcal{TC}_1 = \mathcal{TC} \cup \{\mathsf{tG}_1 \geq \mathsf{tG}_0, \mathsf{tc}_1\}$. The function *isReceivable* checks whether the message ssb[sb[m]] can be decrypted with the keys he has in keys. Every rule has the proviso that to be applicable, the set of comparison constraints and the set of time constraints should be satisfiable.

$\mathsf{tG}_1 \geq \mathsf{tG}_0$ specifying that time can only advance. The rule is fired only if the resulting set of time constraints ($\mathcal{TC} \cup \mathcal{TC}_1$) is consistent, which is done by calling an SMT solver. This way of specifying systems is called Rewriting Modulo SMT [40].

Definition 8. *Let $\mathcal{R}$ be the set of rules in Figure 1. A timed trace is a labeled sequence of transitions written $C_1 \xrightarrow{l_1} C_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{n-1}} C_n$ such that for all $1 \leq i \leq n-1$, $C_i \longrightarrow C_{i+1}$ is an instance of a rule in $\mathcal{R}$ and l_i is $+\mathsf{ms}@\mathsf{tG}_1$ if it is an instance of Send rule sending term ms at time tG_1, $-\mathsf{ms}@\mathsf{tG}_1$ if it is an instance of Receive rule receiving term ms at time tG_1, and $\emptyset$ 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 $\tau = C_1 \xrightarrow{l_1} C_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{n-1}} C_n$ be a trace. For any configuration $C_i = \langle \mathcal{P}_i, \mathcal{IK}_i, \mathcal{DC}_i, \mathcal{EQ}_i, \mathcal{TC}_i\rangle @ \mathsf{tG}_i$, such that $1 \leq i \leq n$, the following holds:*
– *For any $i \leq j \leq n$, $\mathcal{DC}_n(\mathcal{IK}_i)|_{\mathcal{EQ}_n} \subseteq \mathcal{DC}_n(\mathcal{IK}_j)|_{\mathcal{EQ}_n}$, that is, the intruder knowledge can only increase;*

– *Let* sym_k *be a symbol new in some* C_k*,* $k < i$ *and let* sym_i *be a symbol new in* C_i*. If* $\mathsf{dc}(\mathsf{sym}_k, \mathcal{S}_k), \mathsf{dc}(\mathsf{sym}_i, \mathcal{S}_i) \in \mathcal{DC}_i$*, then* $\mathcal{S}_k \subseteq \mathcal{S}_i$*. 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:

$$C_I = \langle \mathcal{P}_I, \mathcal{IK}_I, \mathcal{DC}_I, \mathcal{EQ}_I, \mathcal{TC}_I \rangle @ \mathsf{tG} \text{ and } C'_I = \langle \mathcal{P}'_I, \mathcal{IK}'_I, \mathcal{DC}'_I, \mathcal{EQ}'_I, \mathcal{TC}'_I \rangle @ \mathsf{tG}'$$

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* $\tau = C_1 \xrightarrow{l_1} C_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{n-1}} C_n = \langle \mathcal{P}_n, \mathcal{IK}_n, \mathcal{DC}_n, \mathcal{EQ}_n, \mathcal{TC}_n \rangle @ \mathsf{tG}_n$ *be a timed trace. Its observable is the tuple* $\langle \mathsf{tt}_I, \mathcal{L}_\tau, \mathcal{IK}_n, \mathcal{DC}_n, \mathcal{EQ}_n, \mathcal{TC}_n \rangle$*, where* tt_I *is the global time at configuration* C_1*,* $\mathcal{L}_\tau$ *is the sequence of non-empty labels in* τ*. Let* C *be a configuration. Let* $\mathcal{T}(C)$ *be the set of all traces with initial configuration* C*. The observables of* C *is* $\mathcal{O}(C) = \{O_\tau \mid \tau \in \mathcal{T}(C)\}$*, 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* $O \in \mathcal{O}(C)$ *there exists an equivalent observable* $O' \in \mathcal{O}(C')$*, 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* $O = \langle \mathsf{tt}_I, \mathcal{L}, \mathcal{IK}, \mathcal{DC}, \mathcal{EQ}, \mathcal{TC} \rangle @ \mathsf{tG}$ *and* $O' = \langle \mathsf{tt}'_I, \mathcal{L}', \mathcal{IK}', \mathcal{DC}', \mathcal{EQ}', \mathcal{TC}' \rangle @ \mathsf{tG}'$*, such that*

$$\mathcal{L} = \langle (\pm_1 \mathsf{ms}_1 @ \mathsf{tG}_1) \ldots (\pm_p \mathsf{ms}_p @ \mathsf{tG}_p) \rangle \text{ and } \mathcal{L}' = \langle (\pm'_1 \mathsf{ms}'_1 @ \mathsf{tG}'_1) \ldots (\pm'_n \mathsf{ms}'_n @ \mathsf{tG}'_n) \rangle$$

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,* $\langle \mathsf{ms}_1, \ldots, \mathsf{ms}_N \rangle \sim_{\mathcal{O},\mathcal{O}'} \langle \mathsf{ms}'_1, \ldots, \mathsf{ms}'_N \rangle$*;*
4. *Assume* $\widetilde{\mathsf{tt}}$ *and* $\widetilde{\mathsf{tt}'}$ *are the set of time symbols in* $\mathcal{TC}$ *and* $\mathcal{TC}'$*, respectively. These sets of time symbols are assumed disjoint without loss of generality. The following formulas are tautologies:*

$$\forall \widetilde{\mathsf{tt}}. \left[\mathcal{TC} \Rightarrow \exists \widetilde{\mathsf{tt}'}. \left[\mathcal{TC}' \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N \right] \right]$$
$$\forall \widetilde{\mathsf{tt}'}. \left[\mathcal{TC}' \Rightarrow \exists \widetilde{\mathsf{tt}}. \left[\mathcal{TC} \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N \right] \right]$$

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:

$$\forall \widetilde{\mathsf{tt}}. \left[\mathcal{TC} \Rightarrow \exists \widetilde{\mathsf{tt}'}. \left[\mathcal{TC}' \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N \right] \right]$$

It specifies that for all instances of $\widetilde{\mathsf{tt}}$ that satisfy the constraints $\mathcal{TC}$ in $\mathcal{O}$, it is possible to find instances of $\widetilde{\mathsf{tt}'}$ that satisfy the constraints $\mathcal{TC}'$ in $\mathcal{O}'$, that is, are valid instances, and moreover, the times of the observed messages are identical.

Example Consider the passport protocol role, $\mathcal{P}$, described in Example 5. Moreover, consider the following two initial configurations:

$$C_I = \langle [0 \mid \mathcal{P} \mid \{\mathsf{k}_M, \mathsf{k}_E\}], \{\mathsf{c}_{enc}, \mathsf{c}_{mac}\}, \emptyset, \emptyset, \mathsf{tG} = 0 \rangle @ \mathsf{tG}$$
$$C'_I = \langle [0 \mid \mathcal{P} \mid \{\mathsf{k}_M, \mathsf{k}_E\}], \emptyset, \emptyset, \emptyset, \mathsf{tG}' = 0 \rangle @ \mathsf{tG}'$$

where in C_I the intruder has already eavesdropped a communication between the passport and the identification machine, learning the constants, $\mathsf{c}_{enc} = \mathsf{e}(\mathsf{n}_{old}, \mathsf{k}_E)$ and $\mathsf{c}_{mac} = \mathsf{e}(\mathsf{c}_{enc}, \mathsf{k}_M)$. 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 $\{\mathsf{c}_{enc}, \mathsf{c}_{mac}\}$. 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:

$$\mathcal{L} = \langle (+\mathsf{n}_{new} @ \mathsf{tG}_1), (-\langle \mathsf{sym}_1, \mathsf{sym}_2 \rangle) @ \mathsf{tG}_2), (+error @ \mathsf{tG}_6) \rangle,$$
$$\mathcal{IK} = \{\mathsf{c}_{enc}, \mathsf{c}_{mac}, \mathsf{n}_{new}\}, \quad \mathcal{DC} = \{\mathsf{dc}(\mathsf{sym}_1, \mathcal{IK}), \mathsf{dc}(\mathsf{sym}_2, \mathcal{IK})\},$$
$$\mathcal{EQ} = \{\mathsf{eq}(\mathsf{sym}_2, \mathsf{e}(\mathsf{sym}_1, \mathsf{k}_M)), \mathsf{neq}(\mathsf{sym}_1, \mathsf{e}(\mathsf{n}_{new}, \mathsf{k}_E)\}$$
$$\mathcal{TC} = \{\mathsf{tG}_0 = 0, \mathsf{tt}_0 = \mathsf{tG}_2, \mathsf{tt}_1 = \mathsf{tt}_0 + \mathsf{r}_{mac}, \mathsf{tt}_2 = \mathsf{tt}_1 + \mathsf{r}_{enc}, \mathsf{tG}_6 = \mathsf{tt}_2\} \cup \{\mathsf{tG}_{i+1} \geq \mathsf{tG}_i \mid 0 \leq i \leq 5\}$$

From the time constraints $\mathcal{TC}$, we deduce that $\mathsf{tG}_6 = \mathsf{r}_{mac} + \mathsf{r}_{enc} + \mathsf{tG}_2$.
On the other hand, there is only one trace from C'_I, which yields the observable:

$$\mathcal{L}' = \langle(+\mathsf{n}_{new}@\mathsf{tG}'_1), (-\langle \mathsf{sym}_1, \mathsf{sym}_2\rangle)@\mathsf{tG}'_2), (+error@\mathsf{tG}'_3)\rangle,$$
$$\mathcal{IK}' = \{\mathsf{n}_{new}\}, \quad \mathcal{DC} = \{\mathsf{dc}(\mathsf{sym}'_1, \mathcal{IK}'), \mathsf{dc}(\mathsf{sym}'_2, \mathcal{IK}')\},$$
$$\mathcal{EQ}' = \{\mathsf{neq}(\mathsf{sym}'_2, \mathsf{e}(\mathsf{sym}_1, \mathsf{k}_M))\}$$
$$\mathcal{TC}' = \{\mathsf{tG}'_0 = 0, \mathsf{tt}'_0 = \mathsf{tG}'_2, \mathsf{tt}'_1 = \mathsf{tt}'_0 + \mathsf{r}_{mac}, \mathsf{tG}'_3 = \mathsf{tt}'_1\} \cup \{\mathsf{tG}'_{i+1} \geq \mathsf{tG}'_i \mid 0 \leq i \leq 2\}$$

From the time constraints $\mathcal{TC}'$, we can infer that $\mathsf{tG}'_3 = \mathsf{tG}'_2 + \mathsf{r}_{mac}$.

Clearly, the condition on the time variables for observational equivalence cannot be satisfied, as $\mathsf{tG}_2 = \mathsf{tG}'_2$ and $\mathsf{tG}_6 = \mathsf{tG}'_3$ 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:

$$\forall\widetilde{\mathsf{tt}}.\left[\mathcal{TC} \Rightarrow \exists\widetilde{\mathsf{tt}'}.\left[\mathcal{TC}' \wedge |\mathsf{tG}_1 - \mathsf{tG}'_1| \leq \epsilon \wedge \cdots \wedge |\mathsf{tG}_N = \mathsf{tG}'_N| \leq \epsilon\right]\right]$$
$$\forall\widetilde{\mathsf{tt}'}.\left[\mathcal{TC}' \Rightarrow \exists\widetilde{\mathsf{tt}}.\left[\mathcal{TC} \wedge |\mathsf{tG}_1 - \mathsf{tG}'_1| \leq \epsilon \wedge \cdots \wedge |\mathsf{tG}_N - \mathsf{tG}'_N| \leq \epsilon\right]\right]$$

The greater the value of ϵ, the weaker is the capability of the intruder to measure the timing of observables. For example, if ϵ is the time for encrypting terms, the correspondingly 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 $\exists\forall$:

$$\forall\widetilde{\mathsf{tt}}.\left[\mathcal{TC} \Rightarrow \exists\widetilde{\mathsf{tt}'}.\left[\mathcal{TC}' \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N\right]\right] \text{ is a tautology}$$
$$\Leftrightarrow \neg\forall\widetilde{\mathsf{tt}}.\left[\mathcal{TC} \Rightarrow \exists\widetilde{\mathsf{tt}'}.\left[\mathcal{TC}' \wedge \mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N\right]\right] \text{ is unsat}$$
$$\Leftrightarrow \exists\widetilde{\mathsf{tt}}.\left[\mathcal{TC} \wedge \forall\widetilde{\mathsf{tt}'}.\left[\mathcal{TC}' \Rightarrow \neg\left[\mathsf{tG}_1 = \mathsf{tG}'_1 \wedge \cdots \wedge \mathsf{tG}_N = \mathsf{tG}'_N\right]\right]\right] \text{ is unsat}$$

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 $\exists\forall$ 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.[8] This allows the implementation of

[8] Initially this was implemented using CVC4 [4]. Since Alpha 114 there is also the option to use Yices2.

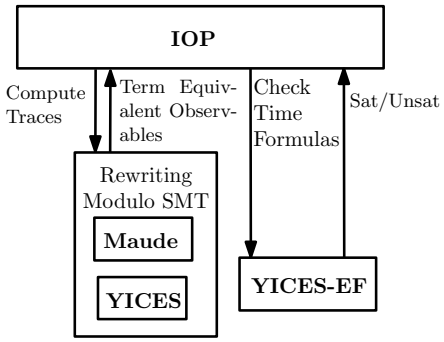

Fig. 2. Timed Observational Equivalence Solver Architecture.

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 *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.

Scenario	Result	Observables	States
Red-Pill	Not Equiv	19/19	74/74
Passport	Not Equiv	36/27	138/112
Passport-Corrected	Equiv	36/27	138/112
Anonymous	Not Equiv	2/3	7/9

Table 1. Experimental Results. Each experiment involves proving the timed observational equivalence of two configurations. It contains number of observables (traces) for each configuration and the total number of states in the whole search tree required to traverse to enumerate all observables.

- **Passport Example**: Consider the timed protocol role specified in Example 5. We checked whether the intruder can distinguish the following two configurations both with two protocol sessions: the first where both protocol sessions are carried out with the same passport and the second where the protocol sessions are carried out with different passports.
- **Corrected Passport Example**: We additionally considered a modification of the Passport example where the timed protocol is corrected in the sense that it sends both error messages at the same time.
- **Anonymous Protocol**: Consider the timed protocol specified in Example 7. We checked whether it is possible for an intruder to distinguish whether two players belong to the same group or not. That is, we checked whether the initial configuration with a player that receives a message from a member of the same group is timed equivalent to the initial configuration with a player that receives a message from a player of a different group.

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 quantificaion 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 ϵ 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.*, $\mathtt{tt}_1 \geq \mathtt{tt}_2 + 10$, 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

1. M. Abadi and C. Fournet. Private authentication. *Theor. Comput. Sci.*, 322(3):427–476, 2004.
2. G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1997.
3. M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In CSF, 2010.
4. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In CAV, 2011.
5. D. Basin and L. V. Sebastian Mödersheim. OFMC: A symbolic model checker for security protocols. *Interational Journal of Information Security*, 2004.
6. G. Bella and L. C. Paulson. Kerberos version 4: Inductive analysis of the secrecy goals. In ESORICS, 1998.
7. N. Benton, M. Hofmann, and V. Nigam. Effect-dependent transformations for concurrent programs. In *PPDP*, 2016.
8. S. Brands and D. Chaum. Distance-bounding protocols (extended abstract). In *EUROCRYPT*, pages 344–359, 1993.
9. I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *CSFW*, pages 55–69, 1999.
10. V. Cheval and V. Cortier. Timing attacks: symbolic framework and proof techniques. In *POST*, 2015.
11. T. Chothia and V. Smirnov. A traceability attack against e-passports. In *FC*, 2010.
12. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework* 2007.
13. R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed model checking of security protocols. In *FMSE*, 2004.
14. V. Cortier and S. Delaune. A method for proving observational equivalence. In *CSF*, 2009.
15. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
16. F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. Built-in variant generation and unification, and their applications in maude 2.7. In *IJCAR*, 2016.
17. B. Dutertre. Yices 2.2. In CAV, 2014.
18. B. Dutertre. Solving exists/forall problems with yices. In *SMT*, 2015.
19. S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: cryptographic protocol analysis modulo equational properties. In *FOSAD*, pages 1–50, 2009.
20. S. Escobar, C. A. Meadows, and J. Meseguer. Maude-npa: Cryptographic protocol analysis modulo equational properties. In *FOSAD Tutorial Lectures*, pages 1–50, 2007.
21. N. Evans and S. Schneider. Analysing time dependent security properties in CSP using PVS. In *ESORICS*, 2000.

22. I. Gazeau and S. Kremer. Automated analysis of equivalence properties for security protocols using else branches. In *ESORICS*, 2017.
23. A. González-Burgueño, D. Aparicio-Sánchez, S. Escobar, C. A. Meadows, and J. Meseguer. Formal verification of the yubikey and yubihsm apis in maude-npa. In *LPAR*, 2018.
24. A. González-Burgueño, S. Santiago, S. Escobar, C. A. Meadows, and J. Meseguer. Analysis of the pkcs#11 API using the maude-npa tool. In *SSR*, 2015.
25. R. Gorrieri, E. Locatelli, and F. Martinelli. A simple language for real-time cryptographic protocol analysis. In *ESOP*, 2003.
26. C. A. Gunter. *Semantics of programming languages - structures and techniques*. Foundations of computing. MIT Press, 1993.
27. G. Ho, D. Boneh, L. Ballard, and N. Provos. Tick tock: Building browser red pills from timing side channels. In *WOOT*, 2014.
28. G. Jakubowska and W. Penczek. Modelling and checking timed authentication of security protocols. *Fundam. Inf.*, 79(3-4):363–378, Aug. 2007.
29. M. I. Kanovich, T. B. Kirigin, V. Nigam, A. Scedrov, and C. L. Talcott. Towards timed models for cyber-physical security protocols. Available in Nigam's homepage, 2014.
30. M. I. Kanovich, T. B. Kirigin, V. Nigam, A. Scedrov, C. L. Talcott, and R. Perovic. A rewriting framework for activities subject to regulations. In *RTA*, 2012.
31. I. A. Mason and C. L. Talcott. IOP: The InterOperability Platform & IMaude: An interactive extension of Maude. In *WRLA*, 2004.
32. C. Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
33. C. A. Meadows. Analysis of the internet key exchange protocol using the NRL protocol analyzer. In *1999 IEEE Symposium on Security and Privacy*, 1999.
34. C. A. Meadows. A cost-based framework for analysis of denial of service networks. *Journal of Computer Security*, 9(1/2):143–164, 2001.
35. C. A. Meadows, R. Poovendran, D. Pavlovic, L. Chang, and P. F. Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*, 2007.
36. R. Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.
37. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.
38. V. Nigam, C. Talcott, and A. A. Urquiza. Symbolic timed observational equivalence. https://arxiv.org/abs/1801.04066, 2018.
39. V. Nigam, C. L. Talcott, and A. A. Urquiza. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In *ESORICS*, 2016.
40. C. Rocha. *Symbolic Reachability Analysis for Rewrite Theories*. PhD thesis, University of Illinois at Urbana-Champagne, 2012.
41. S. Santiago, S. Escobar, C. A. Meadows, and J. Meseguer. A formal definition of protocol indistinguishability and its verification using maude-npa. In *International Workshop on Security and Trust Management*, 2016.