Symbolic Timed Observational Equivalence

Vivek Nigam∗‡ Carolyn Talcott† Abraão Aires Urquiza∗

∗Federal University of Paraíba, João Pessoa, Brazil, {vivek.nigam,abraauc}gmail.com
†SRI International, Menlo Park, USA, clt@csl.sri.com
‡fortiss, Munich, Germany

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 him avoid defense mechanisms, or track a user among others violating his privacy. Observational equivalence is the technical machinery used for verifying whether two systems are distinguishable. Automating the check for observational equivalence suffers the problem of 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 observational 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 preliminary experiments demonstrating the feasibility of our approach.

1. Introduction

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, honeypots are normally used for attracting intruders in order to defend real systems from their attacks. However, as honeypots run over 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, 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 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 observational equivalence have been proposed in the programming languages community as well as in concurrent systems [2], [7], [22], [28] using, for example, logical relations and bisimulation. Observational 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 observational 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 Observational Equivalence: We propose a novel definition of timed equivalence over timed protocol instances [30]. Timing information, e.g., duration of computation, is left symbolically and can be specified in the form of time constraints relating multiple time symbols, e.g., t1 ≥ tt2 + 10;

• SMT Solvers for proving Time Observational Equivalence: SMT solvers are used in two different ways. We specify the operational semantics of timed protocols using Rewriting Modulo SMT [31]. Instead of instantiating time symbols with concrete values, in Rewriting Modulo SMT, a configuration of the system is symbolic and therefore may represent an unbounded number of concrete configurations. Rewriting of a symbolic configuration is only allowed if the set of (time) constraints in the resulting state is satisfiable. SMT-Solvers are used to perform this check. This means not only that there are a finite number of symbolic traces starting from a given configuration, but also reduces considerably the search space needed to enumerate these traces. We demonstrate this with experiments.

The second application of SMT-Solvers is on the proof of timed observational equivalence, namely, to check whether the timing of observations can be matched. This check involves the checking for the satisfiability of \( \exists \forall \) formulas [17].

• Implementation: Relying on the Maude [12] support for Rewriting Modulo SMT using the SMT-solvers CVC4 [4] or Yices [2], we implemented in Maude the machinery necessary for enumerating symbolic traces. However, as
checking for the satisfiability of \( \exists V \) formulas [17] is not supported by Maude, we integrate our Maude machinery with the SMT solver Yices [17]. We carry out some proof-of-concept experiments demonstrating the feasibility of our approach.

Section 2 describes some motivating examples on how intruders can use time side channels for his benefit. We introduce the basic symbolic language in Section 3 and the timed protocol language in Section 4. Section 5 introduces symbolic timed observational 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.

Some missing proofs are shown in the Appendix.

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 [23]. The attack is based on the concept of red pills. The overall goal of the attacker is to determine whether some system is running on a virtual machine or not. 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 [23]. The system running in a virtual machine or a concrete machine follow 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 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, broadcast information intended for passport readers. Also, once powered, e-passport broadcasts can’t be turned off. Chothia and Smirnov identified a flaw in one of the passports 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 fail but the encryption check will succeed, 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 within 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, \textit{i.e.}, in the same shape as \( m' \). In this way, other participants and outsiders cannot determine whether a 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. Term Language

The basic term language contains usual cryptographic operators such as encryption, nonces, tuples. More precisely the term language is defined by the following grammar. We assume given text constants, \( T \) and player names \( P \). We also assume a countable set of nonces, \( N \), and of symbols, \( \text{Sym} \), as well as a countable number of sorted variables, \( \forall V \), where \( N, \text{Sym} \) and \( V \) are disjoint. Below \( v_p \) represents \( p \)-th
a variable of sort player.

Basic Constants:
\[ \text{c := t \in \mathcal{T}} \]
Text Constants
\[ | p \in \mathcal{P} \]
Player Names
\[ | n \in \mathcal{N} \]
Nonces

Keys:
\[ | \text{symk} \]
Symmetric key
\[ | \text{pk}(p) | \text{pk}(v_p) \]
Public key of a player
\[ | \text{sk}(p) | \text{sk}(v_p) \]
Secret key of a player

Symbols:
\[ \text{sym := | sym \in \text{Syms}} \]
Symbol

Terms:
\[ m := | \text{c} \]
Basic constants
\[ | k \]
Keys
\[ | v \in \mathcal{V} \]
Variables
\[ | \text{sym} \in \text{Syms} \]
Symbols
\[ | e(m, k) \]
Encryption of term m with key k
\[ | (m_1, \ldots, m_n) \]
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. \( m, m_1, m_2, \ldots \) will range over symbolic terms. We define \( \text{Syms}(m) \) as the set of symbols appearing in a symbolic term.

It is possible to add other cryptographic constructions, such as hash, signatures, but in order to keep things simple and more understandable, we only include encryption. As hashes and signatures can be specified using encryption, this is not limiting. Finally, it is easy to extend the results here with fresh keys. These are treated in the same way as nonces, but to keep it simple, we do not include them.

We will use two types of (capture avoiding) substitutions. Variable substitutions written \( \text{sb}, \text{sb}_1, \text{sb}_2, \ldots \) which are maps from variables to symbolic terms \( \text{sb} = \{ v_1 \mapsto m_1, v_2 \mapsto m_2, \ldots, v_n \mapsto m_n \} \). Symbolic substitutions written \( \text{ssb}, \text{ssb}_1, \text{ssb}_2, \ldots \) mapping symbols to symbolic terms \( \text{ssb} = \{ \text{sym}_1 \mapsto m_1, \text{sym}_2 \mapsto m_2, \ldots, \text{sym}_n \mapsto m_n \} \).

3.1. Symbolic Term Constraints

Intuitively, variables are entities that can be replaced by symbolic terms, while a symbol denotes a (possibly infinite) set of terms. For example, if the symbol \( \text{sym} \) can be instantiated by any one of the (symbolic) terms \( \{ m_1, \ldots, m_n \} \), then the symbolic term \( e(\text{sym}, k) \) represents the set of terms:

\[ \{ e(m, k) \mid m \in \{ m_1, \ldots, m_n \} \} \]

Such simple idea has enabled the verification of security protocols, which have infinite search space on ground terms, but finite state space using symbolic terms.

We formalize this idea by using derivability constraints. Derivability constraints are constructed over minimal sets defined below.

Definition 3.1. A set of symbolic messages \( S \) is minimal if it satisfies the following conditions:

- \( S \) contains all guessable constants, such as player names and public keys;
- \( S \) does not contain tuples;
- if \( e(m, k) \in S \) and only if \( k^{-1} \notin S \) where \( k^{-1} \) is the inverse key of \( k \);

Formally, the symbolic terms derivable from \( S \) is the smallest set \( \mathcal{R} \) defined inductively as follows:

- if \( m \in S \) then \( m \in \mathcal{R} \);
- if \( k \in \mathcal{R} \) and \( m \in \mathcal{R} \), then \( e(m, k) \in \mathcal{R} \);
- if \( m_1, \ldots, m_n \in \mathcal{R} \), then \( m'_1, \ldots, m'_n \) \( k \) are all guessables, namely player names and public keys.

From two minimal sets, \( S_1 \) and \( S_2 \), we can construct the minimal set, \( S \), representing the union of \( S_1 \) and \( S_2 \) by applying the following operations until a fixed point is reached starting from \( S_0 = S_1 \cup S_2 \):

- if \( m \in S_i \) and \( k^{-1} \notin S_i \) then \( S_{i+1} = S_i \cup \{ m \} \);
- if \( e(m, k), k, k^{-1} \in S_i \), then \( S_{i+1} = S_i \setminus \{ e(m, k) \} \cup \{ m \} \);
- \( \{ m_1, \ldots, m_n \} \in S_i \), then \( S_{i+1} = S_i \setminus \{ m_1, \ldots, m_n \} \cup \{ m'_1, \ldots, m'_n \} \).

For example, given the minimal sets:

\[ S_1 = [\text{symk}, \text{pk}] \] and \( S_2 = [e(\text{e(t, sk), t}), \text{symk}] \)

The minimal set obtained by the union of \( S_1 \) and \( S_2 \) is:

\[ S = [\text{symk}, \text{pk}, \text{t}, \text{e(t, sk)}] \]

We consider two types of constraints on terms: Derivability constraints (Definition 3.2) and comparison constraints (Definition 3.6).

Definition 3.2. A derivability constraint has the form \( \text{dc}(\text{sym}, S) \), where \( S \) is minimal. This constraint denotes that \( \text{sym} \) can be any (symbolic) term derived from \( S \).

For example, the derivability constraint

\[ \text{dc}(\text{sym}, \{ \text{alice, bob, eve}, \text{pk(alice)}, \text{pk(bob)}, \text{sk(eve)} \} ) \]

specifies that \( \text{sym} \) may be instantiated by, e.g., the terms \( \{ \text{alice, bob}, \text{alice, sk(eve)} \} \), \( e(\text{alice, pk(bob)}) \) and \( e(\text{alice, bob}, \text{pk(alice)}, \text{pk(bob)}) \) and so on.

To improve readability (and also reflect our implementation), we will elide in any constraint \( \text{dc}(\text{sym}, S) \) the guessable terms. For example, we write the derivability constraint above simply as \( \text{dc}(\text{sym}(1), \{ \text{sk(eve)} \}) \) as \( \text{alice, bob, eve, pk(alice), pk(bob)} \) are all guessables, namely player names and public keys.

Notice that any \( \text{dc}(\text{sym}, S) \) denotes an infinite number of symbolic terms due to the tupling closure. We will abuse notation and use \( m \in \text{dc}(\text{sym}, \{ m_1, \ldots, m_n \} ) \) to denote that the symbolic term \( m \) is in the set of terms that \( \text{sym} \) can be instantiated with. Moreover, we assume that for any given set of derivability constraints \( \mathcal{DC} \), there is at most one derivability constraint for any given \( \text{sym} \), that is, if \( \text{dc}(\text{sym}, S_1), \text{dc}(\text{sym}, S_2) \in \mathcal{DC} \), then \( S_1 = S_2 \). We write \( \text{dom}(\mathcal{DC}) = \{ \text{sym} \mid \text{dc}(\text{sym}, S) \in \mathcal{DC} \} \). We write \( \text{DC}(\text{sym}) \) for the derivability constraint for \( \text{sym} \) in \( \mathcal{DC} \) if it exists. Moreover, we write \( m \in \text{DC}(\text{sym}) \) if the term can be derived from \( \text{DC}(\text{sym}) \).

Definition 3.3. The symbol dependency graph of a given set of derivability constraints \( \mathcal{DC} \), written \( \mathcal{G}_{\mathcal{DC}} \), is a directed graph defined as follows:
Its nodes are symbols in $DC$, that is, $dom(DC)$; it contains the edge $\text{sym}_1 \rightarrow \text{sym}_2$ if and only if $dc(\text{sym}_1, S_1), dc(\text{sym}_2, S_2) \in DC$ and $S_2$ contains at least one occurrence of $\text{sym}_1$.

While in general the symbol dependency graph of $DC$ can be cyclic, our operational semantics will ensure that these graphs are acyclic.

Consider the following set of derivability constraints:

$$DC_0 = \{ dc(\text{sym}_1, \{ \text{sk}(\text{eve}) \}), dc(\text{sym}_1, \{ e(\text{sym}_1, \text{sym}k) \}), \}
\{ dc(\text{sym}_1, e(\text{sym}_2, \text{sym}_1), pk(\text{Alice})) \},
\{ dc(\text{sym}_4, \{ \text{sym}_3, \text{sym}_2 \}) \}.$$

Its dependency graph is the directed acyclic graph (DAG).

Whenever the dependency graph of a set of constraints is a DAG, we classify the set as acyclic. We can compute a topological sort of the DAG in linear time. For example, a topological sort of $G_{DC}$ is $[\text{sym}_1, \text{sym}_2, \text{sym}_3, \text{sym}_4]$.

Given a set of derivability constraints, we can now formally specify the set of terms that a symbolic term $G$ topological sort of the DAG in linear time. For example, a symbol dependency graph of a set of constraints is $DC$, then $DC$ is the set $\{ \text{sym}_1, \text{sym}_2, \text{sym}_3, \text{sym}_4 \}$.

Let $ms$ be a symbolic term. Let $DC$ be an acyclic set of derivability constraints. Assume $dc(\text{sym}, S) \in DC$. We define the operator $Sub_{DC}(\text{sym}, ms)$ as the set of symbolic terms obtained by replacing all occurrences of $\text{sym}$ in $ms$ by a term $ms_1 \in DC(\text{sym})$. Formally, the set:

$$\{ \sigma(ms) | \sigma = \{ \text{sym} \mapsto ms_1 \} \}$$

is a substitution where $ms_1 \in DC(\text{sym})$.

Moreover, $Sub_{DC}(\text{sym}, S)$ for a set of symbolic terms $S$ is the set $\bigcup_{ms \in S} Sub_{DC}(\text{sym}, ms)$. Let $T = [\text{sym}_1, \ldots, \text{sym}_n]$ be any topological sort of the DAG $G_{DC}$. Then the meaning of a symbolic term $G_{DC}$ with respect to $DC$, written $DC(ms)$, is the set obtained by applying $Sub_{DC}$ consecutively as follows:

$$Sub_{DC}(\text{sym}_1) \cup Sub_{DC}(\text{sym}_2) \cup Sub_{DC}(\text{sym}_3) \cup Sub_{DC}(\text{sym}_4) \cup \ldots \cup Sub_{DC}(\text{sym}_n, ms) \cup \ldots \).$$

For example, $Sub_{DC}(\text{sym}(4), e(\text{sym}(4), pk(\text{bob})))$ is the set of terms:

$$\{ e(\text{ms_1}, pk(\text{bob})) | ms_1 \in DC(\text{sym}(4)) \}$$

It contains the terms $e(\text{sym}_2, pk(\text{bob})), e(\text{sym}_3, pk(\text{bob})), e(\text{sym}_4, pk(\text{bob}))$.

The set $DC_0(e(\text{sym}_1, pk(\text{bob})))$ contains the terms $e(\text{sk}(\text{eve}), \text{sym}k), pk(\text{bob})$, by applying to the term $e(\text{sym}_4, pk(\text{bob}))$ the substitution $[\text{sym}_1 \mapsto \text{sym}_2]$ followed by $[\text{sym}_2 \mapsto e(\text{sym}_1, \text{sym}k)]$.

Notice that for any acyclic set of derivability constraints such that its lowest height symbols (w.r.t. $G_{DC}$) have constraints of the form $dc(\text{sym}, S)$ where $S$ are ground terms, then $DC(ms)$ is an (finite) set of ground terms. This is because the successive application of $Sub_{DC}$ will eventually eliminate all symbols.

Given terms $ms, ms'$, we describe how to check whether $ms \in DC(ms')$. We first build the matching substitution $ssb = [\text{sym}_1' \mapsto ms_1, \ldots, \text{sym}_n' \mapsto ms_n]$ from symbols in $ms'$ to (sub)terms in $ms$. If no such matching substitution exists, then $ms \notin DC(ms')$.

For each $\text{sym}_i' \mapsto ms_i$, let $dc(\text{sym}_i, S_i) \in DC$. We check whether $ms_i \in DC(\text{sym}_i')$ recursively as follows:

- If $ms_i \in S_i$, return true;
- If $ms_1 = e(ms_2, ms_3)$, then we check whether $(ms_2, ms_3) \in DC(\text{sym})$;
- If $ms_1 = \{ ms_1', \ldots, ms_n' \}$, then for each $1 \leq j \leq m$, we check whether $ms_j' \in DC(\text{sym}_i')$.

**Definition 3.5.** $ssb \equiv DC$ if for each symbol $ms \in ssb$, $ms \in DC(\text{sym})$.

The following definitions specify the second type of term constraints called comparison constraints.

**Definition 3.6.** A comparison constraint is either an equality constraint of the form $eq(ms_1, ms_2)$ or an inequality constraint of the form $neq(ms_1, ms_2)$.

A set $EQ$ of comparison constraints should be interpreted as a conjunction of constraints. The following definition specifies when it is satisfiable.

**Definition 3.7.** Let $DC$ be a set of derivability constraints and $EQ$ be a set of comparison constraints. The set $EQ$ is satisfiable w.r.t. $DC$, written $DC \models EQ$, if there is a substitution $\sigma = [\text{sym}_1 \mapsto m_1, \ldots, \text{sym}_n \mapsto m_n]$ mapping all symbols $\text{sym}_i$ in $EQ$ to ground terms in $DC(\text{sym}_i)$, such that:

- for all equality constraints $eq(ms, ms') \in EQ$, $\sigma[ms] = \sigma[ms']$;
- for all inequality constraints $neq(ms, ms') \in EQ$, $\sigma[ms] \neq \sigma[ms']$.

We define the procedure below, $eqCheck$, for checking whether a set of comparison constraints $EQ$ is satisfiable.

**Definition 3.8.** Let $EQ$ be a (finite) set of comparison constraints and $DC$ a set of derivability constraints. Let $eq(ms_1, ms_2), eq(ms_2, ms_3), \ldots eq(ms_{n-1}, ms_n)$ be all the equality constraints in $EQ$. Then $eqCheck(EQ, DC)$ is true if and only if:

1. There is a unifier $ssb$ of the terms $a = (ms_1, \ldots, ms_n)$ and $b = (ms'_1, \ldots, ms'_n)$ mapping symbols to symbolic terms, that is, $ssb(a) = ssb(b)$;
2. For all inequality constraint $neq(ms, ms') \in EQ, ssb[ms] \neq ssb[ms']$;
3. $ssb$ is consistent with $DC$ (as done in Section 3.3).

**Lemma 3.9.** $DC \models EQ$ if and only if $eqCheck(EQ, DC)$.

Moreover, the meaning of a symbolic term should take comparison constraints $EQ$ into account. That is, it should not be possible to replace a symbol by a term that falsifies some comparison constraint. We extend Definition 3.4 accordingly.

**Definition 3.10.** Let $DC$ be an acyclic set of derivability constraints and $EQ$ a set of comparison constraints. The
meaning of a symbolic term $ms$ w.r.t. $DC$ and $EQ$, written $DC(ms)|_{EQ}$, is the set of terms $ms' \in DC(ms)$ such that there exists a matching substitution $\theta$:

- $\theta(ms) = ms'$;
- For all equality constraints $eq(ms_1, ms_2) \in EQ$, $\theta(ms_1) = \theta(ms_2)$;
- For all inequality constraints $neq(ms_1, ms_2) \in EQ$, $\theta(ms_1) \neq \theta(ms_2)$.

For example, $\text{dc}(\text{sym}_1, \{t_1\}), \text{dc}(\text{sym}_2, \{t_2\}) \in DC$ and a set of a single comparison constraint $EQ = \{eq(\text{sym}_1, \text{sym}_2)\}$. The term $\langle t_1, t_2 \rangle \in DC(\langle \text{sym}_1, \text{sym}_2 \rangle)$, but $\langle t_1, t_2 \rangle \notin DC(\langle \text{sym}_1, \text{sym}_2 \rangle)|_{EQ}$. This is because the matching substitution $\theta = \{\text{sym}_1 \mapsto t_1, \text{sym}_2 \mapsto t_2\}$ turns the constraint $\text{eq}(\text{sym}_1, \text{sym}_2)$ false: $t_1 \neq t_2$.

### 3.2. Symbolic Time Constraints

Assume a time signature $\Xi$ which is disjoint to the message alphabet $\Sigma$. It contains numbers (real and natural), variables and pre-defined functions.

- $r_1, r_2, \ldots$: A set of numbers;
- $t_1, t_2, \ldots$: A set of time variables; including the special variable $cur$;
- $+, -, \times, /, \text{floor}, \text{ceiling}, \ldots$: A set of pre-defined functions.

**Time Expressions** are constructed inductively by applying arithmetic symbols to time expressions. For example $\text{ceiling}(2 + t_1 + \text{cur}/10)$ is a Time Expression. The symbols $t_1, t_2, \ldots$ range over Time Expressions. We do not constrain the set of numbers and function symbols in $\Xi$. However, in practice, we allow only the symbols supported by the SMT solver used. All examples in this paper will contain SMT supported symbols (or equivalent). Finally, the variable $cur$ will be a keyword in our protocol specification language denoting the current global time.

**Definition 3.11 (Symbolic Time Constraints).** Let $\Xi$ be a time signature. The set of symbolic time constraints is constructed inductively using time expressions as follows: Let $t_1, t_2$ be time expressions, then

$$t_1 = t_2, \quad t_1 \geq t_2, \quad t_1 > t_2, \quad t_1 < t_2, \quad \text{and} \quad t_1 \leq t_2$$

are Symbolic Time Constraints.

For example, $cur + 10 < \text{floor}(tt - 5)$ is a Time Constraint.

**Time Constraints** will range over $t_1, t_2, t_3, \ldots$.

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 $\{tt_1 \leq 1, tt_2 \geq 3 + tt_3\}$ has an infinite number of models, e.g., $\{tt_1 \mapsto 2.1, tt_2 \mapsto 3.1415\}$.

Finally, SMT-solvers, such as CVC4 [4] and Yices [17], can check for the satisfiability of a set of time constraints.

### 3.3. Symbolic Constraint Solving

For protocol verification, we will assume a traditional Dolev-Yao intruder [15], that is, an intruder that can construct messages from his knowledge by tupling and encrypting messages. However, he cannot decrypt a message for which he does not possess the inverse key. This is captured by the definition of minimal sets Definition 3.1.

**Definition 3.12.** An intruder knowledge $IK$ is a minimal set of symbolic terms.

During protocol execution, the intruder sends messages to honest participants constructed from his knowledge base. 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 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 his current knowledge $IK$.

For example, consider the term $m = \sigma(v_1, \text{sym}, v_1, v_2, k)$ (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 $DC$. We create two fresh symbols $\text{sym}_1$ and $\text{sym}_2$ for, respectively, the variables $v_1$ and $v_2$. We use $sb$ to denote such substitution of variables by symbolic terms. In this example $sb = [v_1 \mapsto \text{sym}_1, v_2 \mapsto \text{sym}_2]$. We then obtain $ms = sb[m] = \sigma(\text{sym}_1, \text{sym}, \text{sym}_1, \text{sym}_2, k)$.

It remains to solve the following problem:

*Given an intruder knowledge, $IK$, and a set of derivability constraints $DC$ constraining the symbols in $IK$, find a representation of all instances of a symbolic term $ms$, satisfying $DC$, that can be generated from $IK$.*

We implemented the function called $\text{sngen}$ that enumerates all possible instances. Its specification is in the Appendix. We describe $\text{sngen}$ informally next and illustrate it with some examples. A similar algorithm is also used by [14].

In particular, $\text{sngen}(m, IK, DC)$ takes as input a term $m$, which is expected by the honest participant, the intruder knowledge $IK$ and the derivability constraints $DC$ for the existing symbols. $\text{sngen}(m, IK, DC)$ then generates as output a pair:

$\{sb, \{ssb_1, DC_1\} \ldots \{ssb_n, DC_n\}\}$

where $sb$ maps the variables of $m$ to symbols, and each $\{ssb_k, DC_k\}$ 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 $m$.

Intuitively, the function $\text{sngen}$ constructs a solution by either matching $m$ with a term in its knowledge $IK$ (base case) or constructing $m$ from terms in $IK$ and using tupling
and encryption. The following examples illustrates the different cases involved:

**Example 3.13.** Consider the following cases for deriving the term \( m = e(v, sym), k \).
- Case 1 (matching with a term in \( IK \)): Assume:
  \[
  IK = \{ e(na, sym_1), k \} \\
  DC = dc(sym, \{ na, n \}) dc(sym_1, S)
  \]
  Then the solution of \( sgen \) is:
  \[
  \{ sb, \{ sym_v \mapsto na, sym \mapsto sym_1 \}, dc(sym_1, add(\{ na, nc \}, S)) \}
  \]
  where \( sb = [v \mapsto sym_v] \) and \( sym_v \) is a fresh symbol. Notice that since \( sym_v \) is mapped to a particular term \( (na, n) \), 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 \( dc(sym_1, S) \).
- Case 2 (constructing terms from \( IK \)): Assume that \( k \in IK \) and \( IK \) has no encryption term. Then the solution of \( sgen \) is:
  \[
  \{ [v \mapsto sym_v], [[], DC] \}
  \]
  which corresponds to generating the term \( e(\{ sym_v, sym \}, k) \).
- Case 3 [No Solution]: Assume that \( IK = \{ e(na, nb), k \} \) and \( DC = dc(sym, \{ na, nc \}) \). Since \( sym \) cannot be instantiated to \( nb \), the intruder cannot use the term \( e(na, nb), k \).

4. Timed Protocol Language

The language used to specify a cryptographic protocol has the standard constructions, such as the creation of fresh values, sending and receiving messages. Moreover, it also includes “if then else” constructors needed to specify, for example, the RFID protocol used by passports. A protocol is composed of a set of roles.

**Definition 4.1 (Timed Protocols).** The set of Timed Protocols, \( TL \), is composed of Timed Protocol Roles, pl, which are constructed by using commands as specified by the following grammar:

\[
\begin{align*}
\text{nil} & \quad \text{Empty Protocol} \\
| \text{(new v # tc), pl} & \quad \text{Fresh Constant} \\
| \text{(+m # tc), pl} & \quad \text{Timed Message Output} \\
| \text{(-m # tc), pl} & \quad \text{Timed Message Input} \\
| \text{(if (m₁ := m₂) # tc then pl₁ else pl₂)} & \quad \text{Timed Conditional}
\end{align*}
\]

Intuitively, \text{new} generates a fresh value binding it to the variable \( v \). \text{(+m # tc)} denotes sending the term \( m \) and the \text{(-m # tc)} receiving a term, and if \( m₁ := m₂ # tc \) then \( pl₁ \) else \( pl₂ \) denotes that if \( m₁ \) can be matched with \( m₂ \), that is, instantiate the variables in \( m₁ \) so that the resulting term is \( m₂ \), then the protocol proceeds by execution \( pl₁ \) and otherwise to \( pl₂ \). A command is only applicable if the associated constraint \( tc \) is satisfiable. We elide the associated time constraint whenever \( tc \) is a tautology, that is, it is always true.

**Example 4.2.** The Needham-Schroeder [29] protocol is specified as follows where \( X, Y \) are variables:

- **Alice** := (new \( N_a \), (+e(\( N_a, alice \), pk(Z))),
  (-e(\( N_b, Y \), pk(alice))), (+e(\( Y, pk(Z) \)));
- **Bob** := (-e(\( X, Z \), pk(bob))), (new \( N_b \),
  (+e(\( X, N_b \), pk(Z))),( -e(\( N_b, pk(bob) \)))

**Example 4.3.** Consider the following protocol role which is a modification of Alice’s role in the Needham-Schroeder’s protocol (Example 4.2):

- **Alice** := (new \( N_a \), (+e(\( N_a, alice \), pk(Z))), (-v, 
  if \( v := e(\( N_a, Y \), pk(alice)) \)
  then (+e(\( Y, pk(Z) \))
  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.

**Example 4.4.** The following role specifies the verifier of a (very simple) distance bounding protocol [8]:

- (new \( v \)), (+v # tt = cur), (-v # 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 \( tt \). Finally, when it receives the response \( v \) it checks whether the current time is less than \( tt + 4 \).

**Example 4.5 (Passport).** Timed conditionals can 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 [11]. For example, consider the following protocol role:

- \text{(new v)}, (+v), (- \{ v_{enc}, v_{mac} \} # tt_0 = cur),
- if \( v_{mac} := e(v_{enc}, k_{mac}) \) # tt = tt_0 + tt_{Mac}
- then \( if (v_{enc} := e(v, k_{enc}) \) # tt = tt_1 + tt_{Enc})
- then (+done # cur = tt_2) else (+error # cur = tt_2)

This role creates a fresh value \( v \) and sends it. Then it is expecting a pair of two messages \( v_{mac} \) and \( v_{enc} \), remembering at time variable \( tt_0 \) when this message is received. It then checks whether the first component \( v_{mac} \) is of the form \( e(v_{enc}, k_{mac}) \), i.e., it is the correct MAC. This operation takes \( tt_{mac} \) time units. The time variable \( tt_1 \) is equal to the time \( tt_0 + tt_{mac} \), 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 exactly 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 \( e(v, k_{enc}) \), which takes (a longer) time \( tt_{Enc} \). If so it sends the done message, otherwise the error message, both at time \( tt_2 \) which is \( tt_1 + tt_{Enc} \).
that sends the differential messages. We assume that it
takes dBase to complete the exchange of the baseline
messages.

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

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 4.7 (Anonymous Protocol).** We specify (a simpli-
fied 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 with the
group key KB. If this is the case, then it checks whether
the player sending the message with key \(v_G\) is part of
the group. If so, then it sends a response encrypted with
his private key. Otherwise, he sends a decoy message.

\[-(\text{(hello)}, e(\text{(hello}, v_0, v_G, \{k_G\})) \# \tt}_0 = \text{cur}\]
\[\text{if } k_G := KB \# \tt}_1 = \tt_0 + dEnc\]
then
\[\text{if } v_G := KA \# \tt}_2 = \tt_1 + dCheck\]
then + \((\text{ack}, e(\text{rsp}, k_B)) \# \tt}_1 + dCreate\]
else + \((\text{ack}, e(\text{decrypt}, k_B)) \# \tt}_1 + dCreate\]
else + \((\text{ack}, e(\text{decrypt}, k_B)) \# \tt}_1 + dCreate\]

Notice the use of time constraints to capture that the
steps of the protocol take some time, namely dCheck
and dCreate.

### 4.1. Operational Semantics for Timed Protocols

The operational semantics of timed protocols is given in
Figure 1. The rewrite rules are rewrite configurations defined
below:

**Definition 4.8.** A symbolic term configuration has the form
\(((P, IK, DC, E\Omega, TC)@tG, where

- \(P\) is a set of player roles of the form \([n | p | \text{keys}]\)
  composed by an identifier, \(n\), a protocol \(p\), and a set of
  known keys \(\text{keys}\);
- \(IK\) is the intruder knowledge;
- \(DC\) is a set of derivability constraints;
- \(E\Omega\) 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
\(m\) which is then added to the intruder knowledge. The
Receive rule expects a term of the form \(m\). The function
\(sgen(m, IK, DC)\) returns the variable substitution \(sb\)
and a set of solutions \(\{ssb, DC\_1\}\) ss. 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 \(m = ssb[ssb[m]]\) 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 \(m\). The rule If-true checks
whether the terms \(m_1\) and \(m_2\) can be matched from the
intruder knowledge \(IK\). This is done by the function sgenB
which is defined in a similar fashion as sgen. It then adds
the equality constraint to the set of comparison constraints.
Finally, the rule If-false replaces the variables in \(m\) by fresh
symbols, constrained in \(DC\_1\) with the intruder knowlegde.
That is if \(sym^2\) is a fresh symbol, then \(dc(sym^2, IK)\). It also
adds the corresponding inequality constraint. The intuition
of replacing variables in \(m_1\) by fresh symbols is to specify
that for any instance of these variables, the resulting term
cannot be matched with \(m_2\) as specifies the inequality
constraint.

**Example 4.9.** Consider the Needham-Schroeder protocol in
Example 4.2. Assume that the intruder initially only knows
his secret key (and the guessables), \(IK_0 = \{sk(\text{eve})\}\) 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((\text{sym}, \text{alice}), pk(\text{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((\text{sym}, \text{sym})), pk(\text{bob})) where
\(\text{sym}_1, \text{sym}_2\) are fresh and constrained \(DC_1 =
\{dc(\text{sym}_1, IK_1), dc(\text{sym}_2, IK_1)\}\). At this point,
Bob creates a fresh value \(N_b\) and sends the message
e((\text{sym}_1 \rightarrow N_b), pk(\text{sym}_2))). The intruder learns this
message (and no further):

\[IK_2 = IK_1 \cup \{e((\text{sym}_1 \rightarrow N_b), pk(\text{sym}_2)))\]\n
Now, the intruder can fool alice by sending her a message
of the form \(e((N_a, Y), pk(\text{alice}))\). We create a fresh
symbol \(\text{sym}_3\) for \(Y\) obtaining \(e((N_a, \text{sym}_3), pk(\text{alice}))\)
and attempt to generate this message from \(IK_2\) us-
ing \(sgen\). Indeed we can generate this message using
e((\text{sym}_1 \rightarrow N_b), pk(\text{sym}_2)) \in IK_2. This generates the
ssb = \{\text{sym}_1 \mapsto N_a, \text{sym}_2 \mapsto alice, \text{sym}_1 \mapsto N_b\}. This
substitution is consistent with \(DC_1\). Notice that \(\text{sym}_3\)
is not constrained. The protocol finishes by the intruder
simply forwarding the message send 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. This can be checked as defined in Definition 3.8.

The second, more interesting, condition is on the time symbols. Whenever a rule is applied, time constraints \( T C_1 \) are added to the configuration’s constraint set. These time constraints are obtained by replacing \( tG \) in \( T C \) with \( tG_N \) together with the constraint \( tG_1 \geq tG_N \) specifying that time can only advance. The rule is fired only if the resulting set of time constraints \( (T C \cup T C_1) \) is consistent, which can be done using SMT solver. This way of specifying systems is called Rewriting Modulo SMT \cite{31}.

**Definition 4.10.** Let \( R \) be the set of rules in Figure 1. A timed trace is a labeled sequence of transitions written \( C_1 \xrightarrow{L_1} C_2 \xrightarrow{L_2} \ldots \xrightarrow{L_{n-1}} C_n \) 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@tG\(_N\) if it is an instance of Send rule sending term \( ms \) at time \( tG_1 \), −ms@tG\(_N\) if it is an instance of Receive rule receiving term \( ms \) at time \( tG_1 \), 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). As observational equivalence involves the matching of traces, checking for observational equivalence can be automated.

**Proposition 4.11.** The set of traces starting from any configuration \( C_0 \) is finite.

**Proposition 4.12.** Let \( \tau = C_1 \xrightarrow{L_1} C_2 \xrightarrow{L_2} \ldots \xrightarrow{L_{n-1}} C_n \) be a trace. For any \( C_i = (P_i, IK_i, DC_i, EQ_i, TC_i) \) such that \( 1 \leq i \leq n \), the following holds:

- For any \( i \leq j \leq n \), \( DC_0(\tau_{IK_i}) + EQ_i \subseteq DC_0(\tau_{IK_j}) + EQ_j \), that is, the intruder knowledge can only increase;
- \( DC_i \) is an acyclic set of derivability constraints;
- Let \( sym_i \) be a symbol created in some \( C_k, k < i \) and let \( sym_i \) be a symbol created in \( C_i \). If \( dc(sym_i, S_k) \in DC_i \) then \( S_k \subseteq S_i \). That is, symbols that are introduced at a 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 \cite{25, 30}. As described in \cite{25}, the standard Dolev-Yao may not be suitable for the verification of Cyber-Physical Security Protocols where the physical properties of the environment is important. Differently from the Dolev-Yao intruder, a timed intruder needs to wait for the message to arrive before he can learn it. \cite{30} proved an upper-bound on the number of timed intruders. Our tool implements this strategy. However, for the examples considered here, the standard Dolev-Yao intruder is enough.

5. **Observational Equivalence**

Our goal now is to determine when two term configurations \( C_i = (P_i, IK_i, DC_i, EQ_i, TC_i)@tG \) and \( C'_i = (P'_i, IK'_i, DC'_i, EQ'_i, TC'_i)@tG' \) cannot be distinguished by the Dolev-Yao intruder. That is, for any trace starting from \( C_i \) there is an equivalent trace starting from \( C'_i \). The following definition specifies observables which collect the necessary information from a trace:

**Definition 5.1.** Let \( \tau = C_1 \xrightarrow{L_1} C_2 \xrightarrow{L_2} \ldots \xrightarrow{L_{n-1}} C_n = (P_n, IK_n, DC_n, EQ_n, TC_n)@tG_n \) be a timed trace. Its observable is the tuple \( \langle tG_1, L'_1, IK'_1, DC'_1, EQ'_1, TC'_1, tG'_n \rangle \), where \( tG_i \) is the global time at configuration \( C_i \), \( L'_i \) 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 \( O(C) = \{ O_i \mid \tau \in \mathcal{T}(C) \} \), that is, the set of all observables of traces starting from \( C \).

Two configurations are observationally equivalent if their observables are equivalent.

**Definition 5.2.** A configuration \( C \) approximates a configuration \( C' \), written \( C \preceq C' \) if for any \( O \in (C) \) there exists an equivalent observable \( O' \in (C') \), that is, \( O \sim O' \) (Definition 5.3). The configurations are observationally equivalent, written \( C \sim C' \), if and only if \( C \preceq C' \) and \( C' \preceq C \).

**Definition 5.3.** Let \( O = \langle L, IK, DC, EQ, TC \rangle@tG \) and \( O' = \langle L', IK', DC', EQ', TC' \rangle@tG' \) be two observables, such that \( L = \{ (\pm ms_1, @tG_1), \ldots, (\pm ms_N, @tG_N) \} \) and \( L' = \{ (\pm ms'_1, @tG'_1), \ldots, (\pm ms'_N, @tG'_N) \} \). 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. There is a black-box bijection \( \text{bij}(O, O') \) between \( O \) and \( O' \) (Definition 5.6);
4. The messages observed are equivalent, that is, \( \langle ms_1, \ldots, ms_N \rangle \sim \langle ms'_1, \ldots, ms'_N \rangle \) (Definition 5.7);
5. Assume \( \tau \) and \( \tau' \) are the set of time symbols in \( TC \) and \( TC' \), respectively. The following formulas are tautologies:

\[ \forall \tau. \left[ TC \Rightarrow \exists \tau'. \left( TC' \land tG_1 = tG'_1 \land \cdots \land tG_N = tG'_N \right) \right] \]
\[ \forall \tau'. \left[ TC' \Rightarrow \exists \tau. \left( TC \land tG_1 = tG'_1 \land \cdots \land tG_N = tG'_N \right) \right] \]

specifying that for any time \( tG_1, \ldots, tG_N \) satisfying \( TC \) when the messages \( ms_1, \ldots, ms_N \) were observed in \( O \), we can find \( tG'_1, \ldots, tG'_N \) satisfying \( TC' \), that makes the time of observation equal.

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. We motivate and define next the conditions 3 and 4 of black-box bijections and term approximation. Finally, the condition 5 specifies that the intruder cannot distinguish the observed messages by measuring the time when they are observed.
New: \langle n | (\text{new } v \# tc), pl \mid \text{keys} \rangle P, IK, DC, EQ, TC) @tg_0 \rightarrow \langle n | sb[pl] \mid \text{keys} \rangle P, IK, DC, EQ, TC_1) @tg_1

where \( n' \) is a fresh nonce and \( sb = [v \mapsto n'] \)

Send: \langle n | (+ms \# tc), pl \mid \text{keys} \rangle P, IK, DC, EQ, TC) @tg_0 \rightarrow \langle n | pl \mid \text{keys} \rangle P, IK \cup \{ms\}, DC, EQ, TC_1) @tg_1

Receive: \langle n | (-\# tc), pl \mid \text{keys} \rangle P, IK, DC, EQ, TC) @tg_0 \rightarrow \\

\text{ssb}([\{n | sb[pl] \mid \text{addKeys}(ms, \text{keys})\}, P, IK, DC_1, EQ, TC_1) @tg_1

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

If-true: \langle n | (if (m_1 := ms_2 \# tc) then \text{pl}_1 else \text{pl}_2) \mid \text{keys} \rangle P, IK, DC, EQ, TC) @tg_0 \rightarrow \\

\text{ssb}([\{n | \text{sb}[pl_1] \mid \text{keys} \}, P, IK, DC_1, EQ \cup \{eq(sb[m_1]), sb[ms_2]\}, TC_1) @tg_1

where \{sb, ssb, DC_1\} css \Rightarrow \text{sgen}(m_1 = ms_2, IK, DC)

If-false: \langle n | (if (m_1 := ms_2 then \text{pl}_1 else \text{pl}_2) \mid \text{keys} \rangle P, IK, DC, EQ) \rightarrow \\

\langle n | \text{sb}[pl_2] \mid \text{keys} \rangle P, IK, DC \cup DC', EQ \cup \{neq(sb[m_1], ms_2)\})

\text{where sb replaces the variables in m_1 by fresh symbols which are constrained with IK in DC}'

Figure 1. Operational semantics for basic protocols. Here \( sb \) is a substitution mapping the variables in \( m \) by fresh symbols; and the function \( mG \) applies the symbol substitution \( sb \) to the range of the variable substitution \( sb; tc \) is the time constraint obtained by replacing cur in tc by the global time \( tg; \) and \( TC = TC \cup \{tg_0, tc_0\} \). The function isReceivable checks whether the message \( sb[pl[m]] \) can be decrypted with the keys he has in \( keys \). Every rule has the proviso that the set of comparison constraints and the set of time constraints should be satisfiable. Rules are only applicable if the set of time constraints is consistent.

5.1. Black-Boxed Terms

There are some subtleties involved in handling nonces and handling encrypted terms for which the intruder does not possess the inverse key. Consider the following sequence of labels:

\[ L_1 = \langle (+n_1), (+n_2), (+n_1, n_2) \rangle \]
\[ L_2 = \langle (+n_1), (+n_2), (+n_2, n_1) \rangle \]

where \( n_1, n_2 \) are nonces, say created by an honest participant. The intruder can distinguish these observables because of the order of the pair of the last send messages. However, the intruder should not be able to distinguish the following two sequence of labels:

\[ L'_1 = \langle (+n_1), (+n_2), (+n_1, n_2) \rangle \]
\[ L'_2 = \langle (+n'_1), (+n'_2), (+n'_2, n'_1) \rangle \]

where \( n_1, n_2, n'_1, n'_2 \) are nonces. This is because although different constants, they appear in the same order. The same happens for encrypted terms in which the intruder cannot decrypt.\(^1\)

Consider the following two sequence of labels:

\[ L''_1 = \langle (+e(t_1, k_1)), (+e(t_2, k_2)), (+e(t_1, k_1), e(t_2, k_2)) \rangle \]
\[ L''_2 = \langle (+e(t'_1, k'_1)), (+e(t'_2, k'_2)), (+e(t'_2, k'_2), e(t'_1, k'_1)) \rangle \]

where \( t_1, t_2 \) are different constants and \( k_1, k_2 \) are keys for which the intruder does not possess the decryption key. Although the intruder cannot decrypt the exchanged messages, he can still distinguish \( L''_1 \) and \( L''_2 \) because of the order of last tuple sent.

The definition below specifies when two observables are equal. It uses definition of black-box bijections and term equivalences defined in the following.

**Definition 5.4.** Let \( O = \langle L, IK, DC, EQ, TC \rangle @tg \) be an observable. The set of black-box terms of \( O \) is the set, written \( BB(O) \), of all sub-terms in \( L \) that are nonces, \( n \), or encryption terms, \( e(t, k) \), for which the inverse of \( k \) is not in \( IK \).

**Definition 5.5.** Let \( O \) be an observable. A symbolic term \( ms \) restricted to \( BB(O) \), written, \( ms|_{BB(O)} \), is the term obtained by replacing the special symbol \( * \) all sub-terms of \( ms \) that (1) do not contain any term in \( BB(O) \) or (2) is not contained in a term in \( BB(O) \).

For example, consider the term \( ms = \langle n_1, t_1, e(t_2, e(t_1, k_2)), k_1 \rangle \) where the intruder knows the inverse of \( k_2 \), but not of \( k_1 \). Then the term restricted to black-boxes is \( \langle n_1, *, e(\langle *, e(t_1, k_2) \rangle, * \rangle \rangle \), where \( t_1, t_2 \) and \( k_1 \) are replaced by \( * \).

**Definition 5.6.** Let \( O = \langle L, IK, DC, EQ, TC \rangle @tg \) and \( O' = \langle L', IK', DC', EQ', TC' \rangle @tg' \) be two observables, such that \( L = \{\langle \pm ms_1, \ldots, \pm ms_n \rangle \} \) and \( L' = \{\langle \pm ms'_1, \ldots, \pm ms'_{n'} \rangle \} \). A black-box bijection, \( bij(O, O') \) is any bijection between the sets \( BB(O) \) and \( BB(O') \) such that \( bij(O, O') \) makes the terms \( \langle ms_1, \ldots, ms_n \rangle|_{BB(O)} \) and \( \langle ms'_1, \ldots, ms'_{n'} \rangle|_{BB(O')} \) equal.\(^2\)

For example, there is no bijection between the terms in \( L''_1 \) and \( L''_2 \) shown above. However, the bijection \( \{e(t_1, k_1) \leftrightarrow e(t'_1, k'_1) \}

\text{and e(t_2, k_2) \leftrightarrow e(t'_2, k'_2) \}} \) makes equal the terms in \( L''_1 \) and the sequence of labels \( L''_2 \) below:

\[ L''_2 = \langle (+e(t'_1, k'_1)), (+e(t'_2, k'_2)), (+e(t'_2, k'_2), e(t'_1, k'_1)) \rangle \]

5.2. Term approximation

**Notation:** For the remainder of this section, to avoid repetition, we will assume given two observables \( O = \langle L, IK, DC, EQ, TC \rangle @tg \) and \( O' =
(L', IK', DC', EQ', TC')@tG' with disjoint sets of term and time symbols and of nonces. Moreover, bij = bij(O, O') is a bijection between the black-box terms of the observables O and O'.

First we handle black boxed in the same way as nonces: given a bijection with n relations:

bij = \{ms_1 \leftrightarrow ms'_1, \ldots, ms_n \leftrightarrow ms'_n\}

We create n new nonces n'_1, \ldots, n'_n and replace each occurrence of ms in O and ms'_i in O' by the same nonce n'_i for 1 \leq i \leq n. In the following we assume that this replacement has been already performed.

We define when a symbolic term approximates another one.

**Definition 5.7.** Let ms and ms' be two symbolic terms in, respectively, O and O'. The term ms approximates ms', written ms \leq_{0,\sigma} ms' if for all ground terms m \in DC \mid_{EQ} (ms), then m \in DC' \mid_{EQ} (ms'). Two terms are equivalent, written ms \sim_{0,\sigma} ms' if and only if ms \leq_{0,\sigma} ms' and ms' \leq_{0,\sigma} ms.

A problem with the definition above is the quantification over all terms, which is an infinite set. We define in Definition 5.12 a procedure, called termEqApprox which checks for the observational equivalence of terms. It uses the auxiliary functions termApprox and canEq.

We start by defining the function termApprox which uses the auxiliary function symDer.

**Definition 5.8.** Let ms and ms' be terms in, respectively, O and O'. The predicate termApprox(ms, ms', O, O') evaluates to true if and only if the two conditions below are satisfied:

1) There exists a matching substitution \( \theta = \{sym'_1 \rightarrow ms_1, \ldots, sym'_n \rightarrow ms_n\} \) mapping symbols in ms' to subterms in ms, such that \( \theta[m] = ms \);

2) For each sym'_i \rightarrow ms_i \in \theta, we have \( \text{symDer}(sym'_i, ms_i, DC, DC') \) (Definition 5.9).

**Definition 5.9.** Let sym' be a symbol in O' and \( \text{dc}(sym', S') \in DC' \) and ms be a term in O. We say that \( \text{symDer}(sym', ms, DC, DC') \) if one of the following holds:

- ms is an guessable;
- ms is a key, then ms \in S';
- ms is a nonce n, then n \in S';
- ms is an encryption e(ms_n, k) \notin BB(O), then \( \text{symDer}(sym', (ms_1, k), DC, DC') \);
- \( ms = (ms_1, \ldots, ms_n) \), then \( \text{symDer}(sym', ms_i, DC, DC') \) for 1 \leq i \leq n;
- ms = sym is a symbol such that \( \text{dc}(sym, S) \in DC \), then for each ms_i \in S, \( \text{symDer}(sym', ms_i, DC, DC') \).

We prove the following soundness and completeness results for the functions symDer and termApprox:

**Theorem 5.10.** Let ms and ms' be two terms. The for all \( m \in DC(ms) \), we have \( m \in DC(ms') \) if and only if termApprox(ms, ms', DC, DC').

**Example:** We illustrate the procedure termApprox with an example. Consider

\[
ms = (n, e((n, t, sym_1, sym_2), k_1))
\]
\[
ms' = (n', e(sym_1, k_1))
\]

Moreover, assume that the intruder knows the inverse of k_1 in both observables, that is, \( k_1^{-1} \in IK \cap IK' \), and consider the bijection \( \text{bij}(O, O') = \{ n \leftrightarrow n' \} \). We replace these nonces by the fresh nonce n'.

We check for Condition 5.8.1, that is, construct a matching substitution mapping symbols in ms' to subterms in ms. We obtain the following matching substitution:

\[
\theta = \{n' \rightarrow (n', t, sym_1, sym_2)\}
\]

as \( \theta[ms] = (n', e((n', t, sym_1, sym_2), k_1)) \).

Notice that if such a matching substitution does not exist, then the ms cannot approximate ms'. Indeed see ms as there is no such matching substitution: The term ms' can be instantiated to a term where a non-tuple is encrypted, e.g., \( (n_1, e(t, k_1)) \) for some guessable t, whereas ms can only be instantiated by terms with at least four elements is encrypted.

Now we check for the Condition 5.8.2. In order to check this condition, we need to know more about DC and DC'. Assume that the symbol \( sym_1 \) is created before the symbol \( sym_2 \) in the trace corresponding to O. Then by Proposition 4.12, we have that \( \text{dc}(sym_1, S_1), \text{dc}(sym_2, S_2) \in DC \), then \( S_1 \subseteq S_2 \).

Consider \( \text{dc}(sym_1, \{t_1, k_1\}), \text{dc}(sym_2, \{t_1, t_2, k_1, k_2\}) \in DC \) which satisfies the condition above. Moreover, assume \( \text{dc}(sym', S') \in DC' \) with \( S' = \{n', t_1, k_1, t_2, k_2\} \).

We have that \( \text{symDer}(sym', \{n, sym_1, sym_2\}, DC, DC') \) as:

- \( \text{symDer}(sym', n', DC, DC') \) as \( n' \in S' \);
- \( \text{symDer}(sym', \{sym_1, DC, DC'\}) \) as \( \{t_1, k_1\} \subseteq S' \);
- \( \text{symDer}(sym', sym_2, DC, DC') \) as \( \{t_1, t_2, k_1, k_2\} \subseteq S' \).

The following definition specifies the function canEq. Intuitively, canEq checks whether it is possible to instantiate w.r.t. DC, EQ the symbols in ms_1, ms_2 so to falsify the constraint neq(ms_1, ms_2).

**Definition 5.11.** canEq(ms_1, ms_2, DC, EQ) evaluates to true if and only if the following conditions are satisfied:

1) If is a unifier \( \sigma = \{sym_1 \rightarrow ms_1, \ldots, sym_n \rightarrow ms_n\} \) of ms_1 and ms_2;
2) \( \sigma \) does not make any comparison constraint in EQ false;
3) For each sym_i \rightarrow ms_i \in \sigma, sym_i can generate the term ms'_i w.r.t. DC.

**Definition 5.12.** Let ms and ms' be terms in, respectively, O and O'. The predicate termEqApprox(ms, ms', O, O') evaluates to true if and only if:

1) If \( \text{eqCheck}(EQ, DC) = \text{false} \);
2) Otherwise if \( \text{eqCheck}(EQ, DC) = \text{eqCheck}(EQ, DC') = \text{true} \), let ssb and ssb' be the corresponding witnessing matching substitutions (Definition 3.8);
a) \text{termApprox}(ssb[ms], ssb'[ms'], ssb[DC], ssb'[DC'])
\text{is true where the witnessing matching substitution is } 
\theta = [\text{sym}_1 \mapsto ms_1, \ldots, \text{sym}_n \mapsto ms_n] \text{ (Definition 5.8); }
\text{b) canEq(ssb[\theta[ms'_1]], ssb[\theta[ms'_2]], DC, EQ) is false}
\text{ (Definition 5.11) for each inequality constraint neq(m'_1, m'_2) \in EQ'.}

Intuitively, the Condition 5.12.1 specifies that if EQ, DC are not consistent, then the approximation is trivial. Otherwise, if both EQ, DC and EQ', DC' are consistent, then there are matching substitutions ssb and ssb'. We apply ssb and ssb' substitutions in, respectively, O and O' in the following. This takes care of all the \text{eq} constraints. Condition 5.12.2a checks whether the terms they generate are the same. Condition 5.12.2b intuitively checks whether it is not possible to falsify any inequality constraint in EQ'
using DC and EQ.

Theorem 5.13. Let ms and ms' be two terms from, respectively, O and O'. Then ms \preceq_{O,O'} ms' if and only if
\text{termEqApprox}(ms, ms', O, O').

Example Consider the protocol role for Alice in Example 4.3. We omit the time constraints as they are not important for this example. From an initial configuration with Alice and the intruder called eve, one can construct the following observable O':

\langle +e((n'_1, alice), pk(eve)), -sym', +error),
IK', DC', \{neq(sym), e(n_1, v), pk(alice))\}\rangle

for some intruder knowledge IK' and DC'.

Consider the protocol role which outputs an error:

Alice' := (new N_d), (+e((N_d, alice), pk(Z))), (-v), (+error)

Consider the observable O for this protocol:

\langle +e((n_1, alice), pk(eve)), -sym, +error), IK, DC, 0)\rangle

which is similar to the observable above, but without the comparison constraint. There is the bijection n_1 \leftrightarrow n'_1.
Moreover, since the message is sent to eve, n_1 \in IK and n'_1 \in IK'.

Notice that
\text{termApprox}(e(n_1, alice), pk(eve)), sym, error), 
(e(n'_1, alice), pk(eve)), sym', error), DC, DC')
as the messages are the same. However, due to the comparison constraint in O, it is not the case that
\text{termEqApprox}(e(n_1, alice), pk(eve)), sym, error), 
(e(n'_1, alice), pk(eve)), sym', error), O, O')

Indeed, the function canEq(sym, e(n_1, b), pk(alice)], DC, 0) is true. (Recall that variables are replaced by fresh symbols.)
This is true because the intruder knows the public key of Alice and n_1.

5.3. Timing of Messages

For Condition 5.3.5, we reduce the formulas to formulas for which existing solvers can be used [17], namely formulas of the form \exists \forall:

\forall t. [T = \exists t'. [\lnot t' \lor t' = t_G] \land \exists t_G \land \lnot t' = t_G] is a tautology
\exists t. [T = \exists t'. [\lnot t' \lor t' = t_G] \land \exists t_G \land \lnot t' = t_G] is unsat
\exists t. [T \land \exists t'. [\lnot t' = t_G] \land \exists t_G \land \lnot t' = t_G] is unsat

In our implementation, we use the SMT solver Yices for solving this formula (which is decidable [17]), where all time variables have type Reals.

Lemma 5.14. Determining whether two timed observables are equivalent is decidable.

Together with Proposition 4.11, we obtain the decidability of timed observational equivalence checking.

Theorem 5.15. Determining whether two configurations are time observationally equivalent is decidable.

6. Experimental Results

We implemented a tool that checks for timed observational equivalence. Its architecture is depicted in Figure 2.

![Figure 2. Timed Observational Equivalence Solver Architecture.](image-url)

[Figure 2: Timed Observational Equivalence Solver Architecture.]

The tool is constructed over the tools:

- **Maude:** We implemented in Maude all the machinery necessary specifying timed protocols as well as checking the term equivalence of observables. Moreover, since its alpha 111 version, Maude allows to call the CVC4 SMT solver [4] from inside Maude programs. This allows the implementation of Rewriting Modulo SMT by using conditional rewrite rules that are only allowed to rewrite if the resulting constraint set is satisfiable. For our applications, this means the set of time constraints;

- **YICES-EF:** Since CVC4 does not provide the API for checking \exists \forall formulas needed for proving time equivalence, we integrated our Maude machinery with the solver YICES-EF which can check for the satisfiability of such formulas. This integration has been carried out by using the IOP framework [27].

Verification is coordinated by the IOP implementation. Given two initial configuration C and C' to be checked for
their timed observational equivalence, IOP sends a command to the Maude+CVC4 tool to enumerate all observables for $C$ and $C'$, computes 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 matching pair, that is, $n = 0$, $C$ and $C'$ are not equivalent. (Similarly for some pair of $C'$.) Otherwise, we continue by the timing equivalence condition. IOP attempts to find at least one time equivalent observable $O'_i$ for $O$. If it does not find it, then $C$ and $C'$ are not equivalent. It does this checking by building the formula as described in Section 5.3 for checking for the timing equivalence of $O$ with $O'_i$. If YICES-EF returns Unsat, they are equivalent. Otherwise they are not equivalent and IOP tries with another observable.

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

- **Red Pill Example**: Consider the timed protocol specified in Example 4.6. 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.

- **Passport Example**: Consider the timed protocol specified in Example 4.5. 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 timing protocol is corrected in the sense that it sends necessarily both error messages at the same time.

- **Anonymous Protocol**: Consider the timed protocol specified in Example 4.7. 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 [30]). This demonstrates the advantage of representing timing symbolically. As expected the number of observable for the passport example were greater as its configurations had two protocol session, 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 some seconds.

### Table 1: Experimental Results

| 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    |

This paper introduced a novel definition of timed equivalence for security protocols using symbolic time constraints. We demonstrated how symbolic time equivalence can be proved automatically with the use of Rewriting Modulo SMT and existing SMT-solvers. The combination of such constraints with Rewriting Modulo SMT greatly reduces the number of states required to enumerate all traces. We implement the machinery for proving the timed observational equivalence and 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 libraries [16] which can be used for the verification of security protocols that use weaker notions of encryption. We are also interested in integrating the machinery developed with Narrowing and Maude-NPA.

**Related Work:** The literature on symbolic verification is vast [5], [9], [14], [14], [18], [20]. However, most of this work uses symbolic reasoning for proving reachability properties.

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 time symbols and branching. Also, 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., $t_1 \geq t_2 + 10$. While a more detailed comparison is left to future work, the use time symbols allows the use of off-the-shelf SMT-solvers which are constantly improving. Indeed, as reported in [10], the corrected version of the Passport Example did not terminate after two days of experiments.

The recent work [20] demonstrates how to automate the prove of observational equivalence of protocols that may...
contain branching and xor. While we allow for branching, we do not consider theories involving xor. However, we do consider timing aspects, which is not considered in [20]. Thus these work 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], [19], [21], [24], [26]. A main difference is that the properties verified were reachability properties and not timed equivalence.

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 Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010, pages 107–121, 2010.
[4] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings, pages 171–177, 2011.
[5] D. Basin and L. V. Sebastian Modersheim. OFMC: A symbolic model checker for security protocols. International Journal of Information Security, 2004.
[6] G. Bella and L. C. Paulson. Kerberos version 4: Inductive analysis of the secrecy goals. In Computer Security - ESORICS 98, 5th European Symposium on Research in Computer Security, Louvain-la-Neuve, Belgium, September 16-18, 1998, Proceedings, pages 361–375, 1998.
[7] N. Benton, M. Hofmann, and V. Nigam. Effect-dependent transformations for concurrent programs. In Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016, pages 188–201, 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 R. Focardi and A. Myers, editors, Proceedings of the 4th International Conference on Principles of Security and Trust (POST’15), Lecture Notes in Computer Science, London, UK, Apr. 2015. Springer Berlin Heidelberg. To appear.
[11] T. Chothia and V. Smirnov. A traceability attack against e-passports. In Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers, pages 20–34, 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, volume 4350 of LNCS. Springer, 2007.
[13] R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed model checking of security protocols. In Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering, FMSE ’04, pages 23–32, New York, NY, USA, 2004. ACM.
[14] V. Cortier and S. Delaune. A method for proving observational equivalence. In Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009, pages 266–276, 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 Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016. Proceedings, pages 183–192, 2016.
[17] B. Dutertre. Solving exists/forall problems with yices. In SMT, 2015.
[18] S. Escobar, C. A. Meadows, and J. Meseguer. Maude-npa: Cryptographic protocol analysis modulo equational properties. In Foundations of Security Analysis and Design V. FOSAD 2007/2008/2009 Tutorial Lectures, pages 1–50, 2007.
Appendix

1. Constraint Solver Algorithm

We define the function $\text{sgen}$ that will compute a finite set of such pairs such that every derivable instance is covered by some pair. In particular, $\text{sgen}$ takes as input:

- $\text{ms}$ – the symbolic term to be generated;
- $\langle I K \rangle$ – the intruder knowledge;
- $\text{ssb}$ – a symbol substitution of symbols to symbolic terms, initially empty ($\theta$). Intuitively, these represent the symbols which are no longer constrained in the set of derivability constraints, but should be replaced by a particular symbolic term;

- $\mathcal{DC}$ – a set of derivability constraints which specify the symbols in $\text{ms}$ and $\langle I K \rangle$ and the range of $\text{ssb}$.

$s\text{gen}(\text{ms}, \langle I K \rangle, \text{ssb}, \mathcal{DC})$ returns a set of solutions: pairs

$$\langle \text{ssb}_1, \mathcal{DC}_1 \rangle, \ldots, \langle \text{ssb}_n, \mathcal{DC}_n \rangle$$

where for $1 \leq i \leq n$, $\text{ssb}_i$ is the symbol substitution with the symbols that have been resolved and $\mathcal{DC}_i$ a set of derivability constraints refining $\mathcal{DC}$.

$s\text{gen}(\text{ms}, \langle I K \rangle, \text{ssb}, \mathcal{DC})$ is defined as follows:

- $\text{ms}$ is a symbol: if $\text{ms} \in \mathcal{DC}$ or $\text{ms}$ has guessable type, then return $\langle \text{ssb}, \mathcal{DC} \rangle$, otherwise the symbol must be constrained, return $\langle \text{ssb} \text{, } \mathcal{DC} \text{d}c(\text{ms}, \langle I K \rangle) \rangle$;

- $\text{ms}$ is a nonce: if $\text{ms} \in \langle I K \rangle$, then return $\langle \text{ssb}, \mathcal{DC} \rangle$, otherwise return the empty set (no solution);

- $\text{ms}$ is a guessable constant, such as, player name, public keys, etc: return $\langle \text{ssb}, \mathcal{DC} \rangle$;

- $\text{ms} = [\text{ms}_1, \ldots, \text{ms}_n]$: iterate through the tuple elements accumulating symbol substitutions and symbol constraints;

$\text{ms} = (\text{ms}_1, \text{ms}_2)$: There are two possibilities:

- The intruder is able to derive $\text{ms}$: This is done by simply calling $\text{sgen}$ on the tuple $[\text{ms}_1, \text{ms}_2]$;

- The intruder possesses an encryption term $\text{ms}_1$ that is unifiable with $\text{ms}$. For this, we proceed in two steps. First, we treat symbols as variables and attempt to unify $\text{ms}$ with terms in $\langle I K \rangle$ checking for cycles (occurences). Once all unifications (symbol substitutions) are found, we only keep the unifications that are consistent with $\mathcal{DC}$. This is done by accumulating the constraints returned by the function $\text{checkSubst}(\text{ssb}_1, \text{ssb}_2, \mathcal{ds})$, defined below, for each unification $\text{ssb}_i$. Note that the domains of $\text{ssb}_1$ and $\text{ssb}_2$ are disjoint.

The function $\text{checkSubst}(\text{ssb}_1, \text{ssb}_2, \mathcal{DC})$ the set of consistent refinements of $\langle \text{ssb}_1, \text{ssb}_2, \mathcal{DC} \rangle$. It processes a binding, $\text{sym} \mapsto \text{ms}_i$, in $\text{ssb}_1$ in the context of a pair $\langle \text{ssb}_2, \mathcal{DC} \rangle$ producing the set of solutions $\mathcal{CSS} = \langle \text{ssb}_2, \mathcal{DC}_2 \rangle$. $1 \leq j \leq k$ that refine $\langle \text{ssb}_2, \mathcal{DC} \rangle$, and bind sym to an instance of ms_i using $\text{chkBnd}(\text{sym}, \text{ms}_i, \text{ssb}_2, \mathcal{DC}_2)$. The next binding is processed in the context of each element of $\mathcal{CSS}$ produced by processing the preceeding bindings.

The function $\text{chkBnd}(\text{sym}, \text{ms}, \text{ssb}, \mathcal{DC})$ works as follows:

- if $\text{sym}$ is not constrained in $\mathcal{DC}$, then $\text{chkBnd}$ returns $\{ (\text{sym} \mapsto \text{ms} \text{ssb}', \mathcal{DC}') \}$ where $\text{ssb}'$ is the result of applying $\text{sym} \mapsto \text{ms}$ to the range of $\text{ssb}$, and $\mathcal{DC}'$ is the result of applying $\text{sym} \mapsto \text{ms}$ to the terms in $\mathcal{DC}$;

- if $\text{sym}$ is constrained in $\mathcal{DC}$, and $\text{ms}$ is a symbol then, then $\text{chkBnd}$ must ensure that $\text{ms}$ is derivable under the accumulated constraints. This can be done by adding the binding $\text{sym} \mapsto \text{ms}$ to each solution returned by $\text{sgen}(\text{ms}, \langle I K \rangle, \text{ssb}_1, \mathcal{DC}_1)$. Here $\mathcal{DC} = \langle I K \rangle$.

In our maude code, it is implemented as the function sGen1 in constraints.maude.

1The freshly symbols are constrained only if necessary, to allow for the possibility that it only appears in an encryption that can be matched in $\langle I K \rangle$.!
\(D_0dc(\text{sym}, IK), \text{ssb}_1\) is the result of applying \(\text{sym} \mapsto m\) to the range of \(\text{ssb}\), and \(D_1\) is the result of applying \(\text{sym} \mapsto m\) to the terms in \(D_0\);

- if \(\text{sym}\) is constrained in \(D\), and \(m\) is a symbol that is not constrained in \(D\) then return \(\{\text{sym} \mapsto m\} \circ \text{ssb}, D_1, dc(\text{sym}, IK)\) where \(\text{ssb}_1, D_1, IK\) are as above.
- if \(\text{sym}\) is constrained in \(D\), and \(m\) is a symbol that is also constrained in \(D\) then return \(\{\text{sym} \mapsto m\} \circ \text{ssb}, D_1, dc(\text{sym}, IK')\) where \(\text{ssb}_1, D_1, IK'\) are as above.

If \(\text{sym}\) is constrained in \(D\), and \(m\) is a symbol that is also constrained in \(D\) then return \(\{\text{sym} \mapsto m\} \circ \text{ssb}, D_1, dc(\text{sym}, IK')\) where \(\text{ssb}_1, D_1, IK'\) are as above.

Intruder knowledge patterns increase over time, thus we are effectively restricting to the lesser of the knowledge sets.

2. Proof of Theorem 5.10

Lemma A.1. Let \(m\) be a term in \(O\) and \(\text{sym}\) a symbol in \(O\). \(m \preceq_{O, \theta} \text{sym}\) if and only if \(\text{sym}Der(\text{sym}', m, DC, DC')\).

Proof: We prove by induction on the greatest height, \(h\), of the symbols in \(m\) in the the dependency graph of the (acyclic) \(D\). In the following assume \(dc(\text{sym}', S') \in D\).

1) Base Case: If \(h = 0\), that is, there are no symbols in \(m\), that is, it is ground. We proceed by induction on \(m\).
   a) Base Case 1: If \(m\) is a guessable, then \(m \preceq_{O, \theta} \text{sym}\) and \(\text{sym}Der(\text{sym}', m, DC, DC')\) are both true;
   b) Base Case 2: If \(m\) is a nonce, then there are two cases. Either, this nonce comes from a bijection, which means that \(m \preceq_{O, \theta} \text{sym}\) if and only if \(n \in DC(\text{sym}')\) if and only if \(n \in S'\) if and only if \(\text{sym}Der(\text{sym}', m, DC, DC')\). Otherwise, \(m \in DC(\text{sym}')\) if and only if \(\text{sym}Der(\text{sym}', m, DC, DC')\);
   c) Inductive Case: If \(m\) is a tuple \((m_1, \ldots, m_n)\), then \(m \preceq_{O, \theta} \text{sym}\) if and only if \(m_i \preceq_{O, \theta} \text{sym}\) if and only if \(\text{sym}Der(\text{sym}', m_i, DC, DC')\) for all \(1 \leq i \leq n\).
   d) Inductive Case: If \(m\) is an encryption \(e(m, k)\), then either it is a black-box and follows the same reasoning as with the nonce case. Otherwise we appeal to the inductive hypothesis as with the tuple case.

2) Inductive Case: If \(h = n + 1\) with \(n \geq 0\), then \(m\) contains some symbols. We proceed by induction on the size of \(m\). Most of the cases have the same reasoning as before in the proof of the base case (Case 1) with the exception of the following base case:
   a) Base Case: If \(m\) is a symbol \(\text{sym}\). Let \(dc(\text{sym}, S)\). All ground terms in \(DC(\text{sym}, S)\) are built using the terms in \(S\). As \(D\) is acyclic, all symbols in \(S\) have height of at most \(n\). Thus we appeal to the IH to prove this case.

Proof: We proceed by induction on the size of \(m\).
   a) Base Case: \(m\) is a symbol \(\text{sym}\). The matching substitution is \(\text{sym} \mapsto m\), which is handled by Lemma A.1.
   b) Other Base Cases: \(m\) is a nonce. Then \(m \preceq_{O, \theta} n\) if and only if \(m = \text{bij}[n']\) if and only if \(\text{term}Approx(m, m', O, O')\). If it is a guessable, then \(m\) has to be the same guessable and \(\text{term}Approx(m, m', O, O')\). Similar when \(m\) is a key.
   c) Inductive Case: \(m \equiv e(m, k)\) is an encrypted term.

3. Proof of Theorem 5.13

Lemma A.2. Let \(EQ = EQ_1 \cup EQ_2\) be a set of comparison constraints, where \(EQ_1\) contains only equality constraints and \(EQ_2\) inequality constraints. Let \(\theta\) be the most general unifier of all constraints in \(EQ_1\). For all ground terms \(m, n \in DC(m)\), \(m \equiv_{\theta} n\) if and only if \(m \equiv_{DC(\theta)} n\).

Proof: \(m \in DC(m)\) if and only if the pattern match \(\text{ssb}\) of \(m\) and \(n\) does not falsify any constraint in \(EQ\) if and only if \(\text{ssb}\) is a set of comparison constraints.

Lemma A.3. There is a ground term \(m \in DC(m)\) and \(m \equiv_{DC(m)} n\) with the same witnessing substitution \(\theta\) if and only if \(\text{can}Eq(m, n, DC, EQ)\).

Proof: \(m \equiv_{DC(m)} n\) and \(m \equiv_{DC(m)} n\) with the same substitution \(\theta\) if and only if \(\theta(m_1) = \theta(m_2)\) if and only if \(m_1 = m_2\) and \(m_2\) is symmetric constraint in \(EQ\) for all \(m, n \in DC\) (Definition of \(DC(\theta)\)) membership if and only if \(\text{can}Eq(m_1, n_1, DC, EQ)\).

Lemma A.4. Let \(EQ\) be a finite set of inequality constraints only. Assume \(DC \vdash EQ\). For all ground terms \(m\) we have that \(m \in DC\) \(\Rightarrow m \in DC\) if and only if \(m \equiv_{DC} m\).

Proof: The reverse direction is immediate. We prove the forward direction. Assume \(m \in DC\) \(\Rightarrow m \in DC\) for all ground terms \(m\) and assume that \(m_1 \in DC\). We show that \(m_1 \in DC\).

We proceed by induction on the size of \(m\). The interesting case is when \(m = \text{sym}\) is a symbol which means that \(m \equiv \text{sym}\) has to be a symbol. Otherwise, it is easy to construct a term \(m_2 \in DC\) such that \(m_2 \equiv \text{sym}\). (For example, a very large tuple of guessables.)

Let the matching symbol substitution \(ssb = [\text{sym} \mapsto m_1]\). There are two cases:

- \(ssb\) does not some \(EQ\) false, then \(m_1 \in DC\) and thus \(m_1 \in DC\) by (1).
- \(ssb\) falsifies some constraint in \(EQ\). Assume by contradiction that \(m_1 \notin DC\). Since \(m_1 \in DC\), an arbitrary large tuple \((m_1, \ldots, m_1) \in DC\). However, since \(m_1 \notin DC\), then \(EQ\) is falsified. (Recall that all constraints in \(EQ\) are inequality constraints.)
constraints.) Then \( \langle m_1, \ldots, m_1 \rangle \in DC(ms)|_{EQ} \). From (1), we get \( \langle m_1, \ldots, m_1 \rangle \in DC'(ms) \) yielding a contradiction with (3). Thus \( m_1 \in DC'(ms) \).

\( \square \)

**Proof:** \( ms \preceq_{O,O'} ms' \) if and only if

- \( DC \not\models EQ \) if and only if \( \text{termEqApprox}(ms, ms', O, O') \) by Lemma 3.9;
- or \( DC \models EQ \) and \( DC' \models EQ' \) and for all \( m \in DC(ms)|_{EQ} \), we have that \( m \in DC'(ms')|_{EQ'} \). By Lemma A.2, we have that \( m \in DC(ms)|_{EQ} \) if and only if \( m \in DC(\theta[ms])|_{EQ_2} \), where \( \theta \) is the mgu of the equality constraints in \( EQ \) and \( EQ_2 \) are the inequality constraints. Similarly \( m \in DC'(ms')|_{EQ'} \) if and only if \( m \in DC'(\theta'[ms'])|_{EQ_2'} \). Thus we only need to consider the inequality constraints \( EQ_2 \) and \( EQ_2' \).

By contraposition, we attempt to find a term \( m \in DC(ms)|_{EQ} \) such that \( m \not\in DC'(ms')|_{EQ'} \). This leads to two possibilities, where \( ssb_1, ssb_1' \) be the symbol substitution such that \( ssb_1[ms] = ssb_1'[ms'] = m \). This can only exists if \( ssb_1[\theta[ms]] = m \) where \( \theta \) is the matching substitution of the symbols in \( ms' \) to terms in \( ms \) (as in Definition 5.8). Otherwise, the terms \( ms \) and \( ms' \) cannot derive the same terms.

- \( ssb_1' \) does not falsify a constraint in \( EQ_2' \) if and only if \( m \not\in DC'(ms') \) if and only if by Lemma A.4 and Theorem 5.10 \( \text{termApprox}(ms, ms', O, O') \) is false;
- \( ssb_1' \) falsifies a constraint in \( \text{neq}(ms_1, ms_2) \in EQ_2' \) if and only if \( ssb_1[\theta(ms_1)] = ssb_1[\theta(ms_2)] \) if and only if \( \text{canEq}(\theta(ms_1), \theta(ms_2), DC, EQ) \) is true (Lemma A.3).

\( \square \)