Timing attacks in security protocols: symbolic framework and proof techniques*

Vincent Cheval[1,2] and Véronique Cortier[1]

[1] LORIA, CNRS, France
[2] School of Computing, University of Kent, UK

Abstract. We propose a framework for timing attacks, based on (a variant of) the applied-pi calculus. Since many privacy properties, as well as strong secrecy and game-based security properties, are stated as process equivalences, we focus on (time) trace equivalence. We show that actually, considering timing attacks does not add any complexity: time trace equivalence can be reduced to length trace equivalence, where the attacker no longer has access to execution times but can still compare the length of messages. We therefore deduce from a previous decidability result for length equivalence that time trace equivalence is decidable for bounded processes and the standard cryptographic primitives.
As an application, we study several protocols that aim for privacy. In particular, we (automatically) detect an existing timing attack against the biometric passport and new timing attacks against the Private Authentication protocol.

1 Introduction

Symbolic models as well as cryptographic models aim at providing high and strong guarantees when designing security protocols. However, it is well known that these models do not capture all types of attacks. In particular, most of them do not detect *side-channel* attacks, which are attacks based on a fine analysis of *e.g.*, time latencies, power consumption, or even acoustic emanations [34,12]. The issue of side-channel attacks is well-known in cryptography. Efficient implementations of secure cryptographic schemes may be broken by a fine observation of the computation time or the power consumption. Of course, counter-measures have been proposed but many variations of side-channel attacks are still regularly discovered against existing implementations.

The same kind of issues occur at the protocol level as well. For example, the biometric passport contains an RFID chip that stores sensitive information such as the name, nationality, date of birth, etc. To protect users' privacy, data are never sent in the clear. Instead, dedicated protocols ensure that confidential data are sent encrypted between the passport and the reader. However, a minor variation in the implementation of the protocol in the French passport has led to a privacy flaw [9]. Indeed, by observing the error message when replaying some old message, an attacker could learn whether a given passport belongs to Alice or not. The attack has been fixed by unifying the error messages produced by the passports. However, it has been discovered [25] that *all*

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 258865, project ProSecure

biometric passports (from all countries) actually suffer from exactly the same attack as soon as the attacker measures the computation time of the passport instead of simply looking at the error messages.

The goal of the paper is to provide a symbolic framework and proof techniques for the detection of timing attacks on security protocols. Symbolic models for security protocols typically assume "the perfect encryption hypothesis", abstracting away the implementation of the primitives. We proceed similarly in our approach, assuming a perfect implementation of the primitives w.r.t. timing. It is well known that implementation robust against side-channel attacks should, at the very least, be "in constant time", that is, the execution time should only depend on the number of blocks that need to be processed. "Constant time" is not sufficient to guarantee against timing attacks but is considered to be a minimal requirement and there is an abundant literature on how to design such implementations (see for example the NaCl library [1] and some related publications [33,16]). One could think that side-channel attacks are only due to a non robust implementation of the primitives and that it is therefore enough to analyze in isolation each of the cryptographic operations. However, in the same way that it is well known that the perfect encryption assumption does not prevent flaws in protocols, a perfect implementation of the primitives does not prevent side-channel attacks. This is exemplified by the timing attack found against the biometric passport [25] and the timing attacks we discovered against the Private Authentication protocol [7] and several of its variants. These attacks require both an interaction with the protocol and a dedicated time analysis. Robust primitives would not prevent these attacks.

Our first contribution is to propose a symbolic framework that models timing attacks at the protocol level. More precisely, our model is based on the applied-pi calculus [4]. We equip each function symbol with an associated time function as well as a length function. Indeed, assuming a perfect implementation of the primitives, the computation time of a function typically only depends on the size of its arguments. Each time a process (typically a machine) performs an observable action (*e.g.*, it sends out a message), the attacker may observe the elapsed time. Our model is rather general since it inherits the generality of the applied-pi calculus with *e.g.*, arbitrary cryptographic primitives (that can be modeled through rewrite systems), possibly arbitrarily replicated processes, etc. Our time and length functions are also arbitrary functions that may depend on the machine on which they are run. Indeed, a biometric passport is typically much slower than a server. Moreover, a server usually handles thousands of requests at the same time, which prevents from a fine observation of its computation time. Our model is flexible enough to cover all these scenarios. Finally, our model covers more than just timing attacks. Indeed, our time functions not only model execution times but also any kind of information that can be leaked by the execution, such as power consumption or other "side-channel" measurements.

Our second main contribution is to provide techniques to decide (time) process equivalence in our framework. Equivalence-based properties are at the heart of many security properties such as privacy properties [29,9] (*e.g.*, anonymity, unlinkability, or ballot privacy), strong secrecy [19] (*i.e.* indistinguishability from random), or game-based security definitions [5,27] (*e.g.*, indistinguishability from an ideal protocol). Side channel attacks are particularly relevant in this context where the attacker typically tries

to distinguish between two scenarios since any kind of information could help to make a distinction. Several definitions of equivalence have been proposed such as trace equivalence [4], observational equivalence [4], or diff-equivalence [18]. In this paper, we focus on trace equivalence. In an earlier work [24], we introduced length (trace) equivalence. It reflects the ability for an attacker to measure the length of a message but it does not let him access to any information on the internal computations of the processes.
Our key result is a generic and simple simplification result: time equivalence can be reduced to length equivalence. More precisely, we provide a general transformation such that two processes P and Q are in time equivalence if and only if their transformation $\tilde{P}$ and $\tilde{Q}$ are in length equivalence, that is $P \approx_{\mathsf{ti}} Q \Leftrightarrow \tilde{P} \approx_{\ell} \tilde{Q}$. This result holds for an arbitrary signature and rewriting system, for arbitrary processes - including replicated processes, and for arbitrary length and time functions. The first intuitive idea of the reduction is simple: we add to each output a term whose length encodes the time needed for the intermediate computations. The time elapsed between two outputs of the same process however does not only depend on the time needed to compute the sent term and the corresponding intermediate checks. Indeed, other processes may run in parallel on the same machine (in particular other ongoing sessions). Moreover, the evaluation of a term may fail (for example if a decryption is attempted with a wrong key). Since we consider else branches, this means that an else branch may be chosen after a failed evaluation of a term, which execution time has to be measured precisely. The proof of our result therefore involves a precise encoding of these behaviors.

A direct consequence of our result is that we can inherit existing decidability results for length equivalence. In particular, we deduce from [24] that time equivalence is decidable for bounded processes and a fixed signature that captures all standard cryptographic primitives. We also slightly extend the result of [24] to cope with polynomial length functions instead of linear functions.

As an application, we study three protocols that aim for privacy in different application contexts: the private authentication protocol (PA) [7], the Basic Authentication Protocol (BAC) of the biometric passport [2], and the 3G AKA mobile telephony protocol [10]. Using the APTE tool [22] dedicated to (length) trace equivalence, we retrieve the flaw of the biometric passport mentioned earlier. We demonstrate that the PA protocol is actually not private if the attacker can measure execution times. Interestingly, several natural fixes still do not ensure privacy. Finally, we provide a fix for this protocol and (automatically) prove privacy. Similarly, we retrieve the existing flaw on the 3G AKA protocol.

Related work. Several symbolic frameworks already include a notion of time [15,30,26,31,32]. The goal of these frameworks is to model timestamps. The system is given a global clock, actions take some number of "ticks", and participants may compare time values. Depending on the approach, some frameworks (e.g. [15,30]) are analysed using interactive theorem provers, while some others (e.g. [26,32]) can be analysed automatically using for example time automata techniques [32]. Compared to our approach, the representation of time is coarser: each action takes a fixed time which does not depend on the received data while the attack on e.g. the biometric passport precisely requires to measure (and compare) the time of a given action. Moreover, these frameworks consider trace properties only and do not apply to equivalence properties.

They can therefore not be applied to side-channel analysis.
On the other hand, the detection or even the quantification of information possibly leaked by side-channels is a subject thoroughly studied in the last years (see e.g. [35,13,37,17,11]). The models for quantifying information leakage are typically closer to the implementation level, with a precise description of the control flow of the program. They often provide techniques to *measure* the amount of information that is leaked. However, most of these frameworks typically do not model the cryptographic primitives that security protocols may employ. Messages are instead abstracted by atomic data. [35] does consider primitives abstracted by functions but the framework is dedicated to measure the information leakage of some functions and does not apply to the protocol level. This kind of approaches can therefore not be applied to protocols such as BAC or PA (or when they may apply, they would declare the flawed and fixed variants equally insecure).
Fewer papers do consider the detection of side-channel attacks for programs that include cryptography [36,8]. Compared to our approach, their model is closer to the implementation since it details the implementation of the cryptographic primitives. To do so, they over-approximate the ability of an attacker by letting him observe the control flow of the program, e.g. letting him observe whether a process is entering a then or an else branch. However privacy in many protocols (in particular for the BAC and PA) precisely relies on the inability for an attacker to detect whether a process is entering a then (meaning e.g. that the identity is valid) or an else branch (meaning e.g. that the identity is invalid). So the approach developed in [36,8] could not prove secure the fixed variants of BAC and PA. Their side-channel analysis is also not automated, due to the expressivity of their framework.

2 Messages and computation time

2.1 Terms

As usual, messages are modeled by terms. Given a *signature* $\mathcal{F}$ (*i.e.* a finite set of function symbols, with a given arity), an infinite set of *names* $\mathcal{N}$, and an infinite set of variables $\mathcal{X}$, the set of terms $\mathcal{T}(\mathcal{F}, \mathcal{N}, \mathcal{X})$ is defined as the union of names $\mathcal{N}$, variables $\mathcal{X}$, and function symbols of $\mathcal{F}$ applied to other terms. In the spirit of [6], we split $\mathcal{F}$ into two distinct subsets $\mathcal{F}_d$ and $\mathcal{F}_c$. $\mathcal{F}_d$ represents the set of *destructors* whereas $\mathcal{F}_c$ represents the set of *constructors*. We say that a term t is a *constructor term* if t does not contain destructor function symbol, *i.e.* $t \in \mathcal{T}(\mathcal{F}_c, \mathcal{N}, \mathcal{X})$. Intuitively, constructors stand for cryptographic primitives such as encryption or signatures, while destructors are operations performed on primitives like decryption or validity checks.

A term is said to be ground if it contains no variable. The set of ground terms may be denoted by $\mathcal{T}(\mathcal{F}, \mathcal{N})$ instead of $\mathcal{T}(\mathcal{F}, \mathcal{N}, \emptyset)$. The set of names of a term M is denoted by $names(M)$. $\tilde{n}$ denotes a set of names. Substitutions are replacement of variables by terms and are denoted by $\theta = \{^{M_1}/_{x_1}, \ldots, ^{M_k}/_{x_k}\}$. The application of a substitution θ to a term M is defined as usual and is denoted $M\theta$. The set of subterms of a term t is denoted $st(t)$. Given a term t and a position p, the subterm of t at position p is denoted $t|_p$. Moreover, given a term r, we denote by $t[r]_p$ the term t where its original subterm at position p is replaced by r.

Example 1. A signature for modelling the standard cryptographic primitives (symmetric and asymmetric encryption, concatenation, signatures, and hash) is $\mathcal{F}_{\mathsf{stand}} = \mathcal{F}_c \cup \mathcal{F}_d$ where $\mathcal{F}_c$ and $\mathcal{F}_d$ are defined as follows (the second argument being the arity):

$$\mathcal{F}_c = \{\mathsf{senc}/2,\ \mathsf{aenc}/2,\ \mathsf{pk}/1,\ \mathsf{sign}/2,\ \mathsf{vk}/1,\ \langle\,\rangle/2,\ \mathsf{h}/1\}$$
$$\mathcal{F}_d = \{\mathsf{sdec}/2, \mathsf{adec}/2, \mathsf{check}/2, \mathsf{proj}_1/1, \mathsf{proj}_2/1, \mathsf{equals}/2\}$$

The function aenc (resp. senc) represents asymmetric (resp. symmetric) encryption with corresponding decryption function adec (resp. sdec) and public key pk. Concatenation is represented by $\langle\,\rangle$ with associated projectors proj_1 and proj_2. Signature is modeled by the function sign with corresponding validity check check and verification key vk. h represents the hash function. The operator equals models equality tests. These tests are typically hard-coded in main frameworks but we need here to model precisely the time needed to perform an equality test.

2.2 Rewriting systems

The properties of the cryptographic primitives (e.g. decrypting an encrypted message yields the message in clear) are expressed through rewriting rules. Formally, we equip the term algebra with a *rewriting system*, that is a set $\mathcal{R}$ of *rewrite rules* $\ell \to r$ such that $\ell \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \setminus \mathcal{X}$ and $r \in \mathcal{T}(\mathcal{F}, vars(\ell))$. A term s is rewritten into t by a rewriting system $\mathcal{R}$, denoted $s \to_{\mathcal{R}} t$ if there exists a rewrite rule $\ell \to r \in \mathcal{R}$, a position p of s and a substitution σ such that $s|_p = \ell\sigma$ and $t = s[r\sigma]_p$. The reflexive transitive closure of $\to_{\mathcal{R}}$ is denoted by $\to^*_{\mathcal{R}}$.

A rewriting system $\mathcal{R}$ is *confluent* if for all terms s, u, v such that $s \to^*_{\mathcal{R}} u$ and $s \to^*_{\mathcal{R}} v$, there exists a term t such that $u \to^*_{\mathcal{R}} t$ and $v \to^*_{\mathcal{R}} t$. Moreover, we say that $\mathcal{R}$ is *convergent* if $\mathcal{R}$ is confluent and terminates.

A term t is in *normal form* (w.r.t. a rewrite system $\mathcal{R}$) if there is no term s such that $t \to_{\mathcal{R}} s$. Moreover, if $t \to^*_{\mathcal{R}} s$ and s is in normal form then we say that s is a normal form of t. In what follows, we consider only convergent rewriting system $\mathcal{R}$. Thus the normal form of a term t is unique and is denoted $t{\downarrow}$.

Example 2. We associate to the signature $\mathcal{F}_{\mathsf{stand}}$ of Example 1 the following rewriting system:

$$\begin{array}{rrl} \mathsf{sdec}(\mathsf{senc}(x,y),y) \to x & \mathsf{check}(\mathsf{sign}(x,y),\mathsf{vk}(y)) \to x & \mathsf{proj}_1(\langle x,y\rangle) \to x \\ \mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) \to x & \mathsf{equals}(x,x) \to x & \mathsf{proj}_2(\langle x,y\rangle) \to y \end{array}$$

The two first rewriting rules on the left represent respectively symmetric and asymmetric encryption. The first two rules on the right represent the left and right projections. The rewriting rule $\mathsf{check}(\mathsf{sign}(x,y),\mathsf{vk}(y)) \to x$ models the verification of signature: if the verification succeeds, it returns the message that has been signed. Finally, the equality test succeeds only if both messages are identical and returns one of the two messages.

A ground term u is called a *message*, denoted $\mathsf{Message}(u)$, if $v{\downarrow}$ is a constructor term for all $v \in st(u)$. For instance, the terms $\mathsf{sdec}(a,b)$, $\mathsf{proj}_1(\langle a, \mathsf{sdec}(a,b)\rangle)$, and $\mathsf{proj}_1(a)$ are not messages. Intuitively, we view terms as modus operandi to compute bitstrings where we use the call-by-value evaluation strategy.

2.3 Length and time functions

We assume a perfect implementation of primitives and we aim at detecting side-channel attacks at the protocol level. In standard robust implementations of encryption, the time for encrypting is constant, that is, it does not depend on the value of the key nor the value of the message but only on the number of blocks that need to be processed. So the computation time of a function depends solely on the length of its arguments. For example, assuming the size of m and k to be a multiple of the size of one block, the time needed to compute $\mathsf{senc}(m, k)$, the encryption of the message m over the key k, depends on the lengths of m and k. We thus introduce time functions as well as length functions.

Length function For any primitive $\mathsf{f} \in \mathcal{F}$ of arity n, we associate a length function from $\mathbb{N}^n$ to $\mathbb{N}$. Typically, the length function of f indicates the length of the message obtained after application of f, based on the length of its arguments. Given a signature $\mathcal{F}$ and a set of length functions L associated to $\mathcal{F}$, we denote by $\mathsf{len}_L^{\mathsf{f}}$ the length function in L associated to f. Moreover we consider that names can have different sizes. Indeed, an attacker can always create a bitstring of any size. Hence we consider an infinite partition of $\mathcal{N}$ such that $\mathcal{N} = \cup_{i \in \mathbb{N}} \mathcal{N}_i$ and each $\mathcal{N}_i$ is an infinite set of names of size i. To ease the reading, we may denote by n^i a name of $\mathcal{N}_i$.

The length of a closed message t, denoted $\mathsf{len}_L(t)$, is defined as follows:

$$\begin{array}{ll} \mathsf{len}_L(n^i) = i & \text{when } n^i \in \mathcal{N}_i \\ \mathsf{len}_L(\mathsf{f}(t_1, \ldots, t_k)) = \mathsf{len}_L^{\mathsf{f}}(\mathsf{len}_L(t_1), \ldots, \mathsf{len}_L(t_k)) & \end{array}$$

We say that a set of length functions L is polynomial if for all $\mathsf{f} \in \mathcal{F}$, there exists a polynomial $P \in \mathbb{N}[X_1, \ldots, X_n]$ (*i.e.* a polynomial of n variables, with coefficients in $\mathbb{N}$) such that for all $x_1, \ldots, x_n \in \mathbb{N}$, $\mathsf{len}_L^{\mathsf{f}}(x_1, \ldots, x_n) = P(x_1, \ldots, x_n)$. The class of polynomial time functions is useful to obtain decidability of (timed) trace equivalence. A particular case of polynomial length functions are *linear* length functions, for which the associated polynomial is linear. Note that the linear length functions are so far the only functions that have been proved sound w.r.t. symbolic models [27].

Example 3. An example of set of length functions L associated to the signature $\mathcal{F}_c$ of Example 1 is defined as follows.

$$\begin{array}{lll} \mathsf{len}_L^{\mathsf{senc}}(x, y) = x & \mathsf{len}_L^{\mathsf{aenc}}(x, y) = x + y & \mathsf{len}_L^{\mathsf{pk}}(x) = x \\ \mathsf{len}_L^{\langle\,\rangle}(x, y) = 1 + x + y & \mathsf{len}_L^{\mathsf{sign}}(x, y) = x + y & \mathsf{len}_L^{\mathsf{vk}}(x) = x \end{array}$$

In this example, the length of a encrypted message is linear in the size of the original message and the length of the key. The concatenation of two messages is of length the sum of the lengths of its arguments, plus some constant size used to code the frontier between the two messages. Note that these length functions are polynomial and even linear. These length functions are rather simple and abstract away some implementation details such as padding but more complex functions may be considered if desired.

Time function For each primitive $\mathsf{f} \in \mathcal{F}$ of arity n, we associate a *time function* from $\mathbb{N}^n$ to $\mathbb{N}$. Given a set of time functions T, we denote $\mathsf{time}^{\mathsf{f}}_T$ the time function associated to f in T. Intuitively, $\mathsf{time}^{\mathsf{f}}_T(x_1, \ldots, x_n)$ determines the computation time of the application of f on some terms $u_1, \ldots, u_n$ assuming that the terms u_i are already computed and the length of u_i is x_i. Finally, we define a constant function modelling the computation time to access data such as the content of a variable in the memory, usually denoted $\mathsf{time}^{\mathcal{X}}_T$.

Example 4. Coming back to the signature $\mathcal{F}_{\mathsf{stand}}$ of Example 1, we can define the set T of time functions as follows:

$$\mathsf{time}^{\mathcal{X}}_T = 1 \quad \mathsf{time}^{\mathsf{proj}_2}_T(x) = 1 \quad \mathsf{time}^{\mathsf{proj}_1}_T(x) = 1 \quad \mathsf{time}^{\langle\rangle}_T(x, y) = 1$$
$$\mathsf{time}^{\mathsf{adec}}_T(x, y) = x \qquad \mathsf{time}^{\mathsf{aenc}}_T(x, y) = x \qquad \mathsf{time}^{\mathsf{equals}}_T(x, y) = x + y$$

In this example, concatenation and projections have constant computation time (e.g., concatenation and projections are done by adding or removing a symbolic link). The asymetric encryption of m by k linearly depends on the size of m. We ignore here the complexity due to the size of the key since key size is usually fixed in protocols. Note it would be easy to add a dependency. Finally the time for an equality test is the sum of the length of its arguments. This corresponds to a naive implementation. We could also choose $\mathsf{time}^{\mathsf{equals}}_T(x, y) = \max(x, y)$. Our framework does not allow to model efficient implementations where the program stops as soon as one bit differs. However, such efficient implementations leak information about the data tested for equality and are therefore not good candidates for an implementation robust against side-channel attacks. Again, other time functions may of course be considered.

The computation time of a term is defined by applying recursively each corresponding time function. More generally, we define the computation time of a term $t\sigma$ assuming that the terms in σ are already computed.

Definition 1. *Let $\mathcal{F}$ be a signature, let L be a set of length functions for $\mathcal{F}$ and let T be a set of time functions for $\mathcal{F}$. Consider a substitution σ from variables to ground constructor terms. For all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{N}, \mathcal{X})$ such that $vars(t) \subseteq \mathrm{dom}(\sigma)$, we define the computation time of t under the substitution σ and under the sets L and T, denoted $\mathsf{ctime}_{L,T}(t, \sigma)$, as follows:*

$$\mathsf{ctime}_{L,T}(t, \sigma) = \mathsf{time}^{\mathcal{X}}_T \qquad \text{if } t \in \mathcal{X} \cup \mathcal{N}$$
$$\mathsf{ctime}_{L,T}(\mathsf{f}(u_1, \ldots, u_n), \sigma) = \mathsf{time}^{\mathsf{f}}_T(\ell_1, \ldots, \ell_n) + \textstyle\sum_{i=1}^{n} \mathsf{ctime}_{L,T}(u_i, \sigma)$$
$$\text{if } \ell_i = \mathsf{len}_L((u_i\sigma)\!\downarrow) \text{ and } \mathsf{Message}(u_i\sigma) \text{ is true } \forall i \in \{1, \ldots, n\}$$
$$\mathsf{ctime}_{L,T}(\mathsf{f}(u_1, \ldots, u_n), \sigma) = \textstyle\sum_{i=1}^{k} \mathsf{ctime}_{L,T}(u_i, \sigma)$$
$$\text{if } \mathsf{Message}(u_i\sigma) \text{ is true } \forall i \in \{1, \ldots, k-1\} \text{ and } \mathsf{Message}(u_k\sigma) \text{ is false}$$

Intuitively, $\mathsf{ctime}_{L,T}(t, \sigma)$ represents the time needed to compute $t\sigma\!\downarrow$ when the terms of σ are already computed and stored in some memory. Therefore the computation time of a variable represents in fact the access time to the memory. We assume in this paper that all primitives are computed using the call-by-value evaluation strategy with a lazy evaluation when failure arises. Hence, when computing $\mathsf{f}(u_1, \ldots, u_n)$ with

the memory σ, the terms u_i are computed first from left to right. If all computations succeed then the primitive f is applied. In such a case, we obtain the computation time $\mathsf{time}^{\mathsf{f}}_T(\mathsf{len}_L(u_1\sigma\downarrow), \ldots, \mathsf{len}_L(u_n\sigma\downarrow)) + \sum_{i=1}^{n} \mathsf{ctime}_{L,T}(u_i, \sigma)$. Otherwise, the computation of $\mathsf{f}(u_1, \ldots, u_n)$ stops at the first u_k that does not produce a message. This yields the computation time $\sum_{i=1}^{k} \mathsf{ctime}_{L,T}(u_i, \sigma)$. We assume here that names are already generated to avoid counting their generation twice. Hence the associated computation time is also $\mathsf{time}^{\mathcal{X}}_T$ the access time to the memory. We will see later in this section how the computation time for the generation of names is counted, when defining the semantics of processes.

3 Processes

Protocols are modeled through processes, an abstract small programming language. Our calculus is inspired from the applied-pi calculus [4].

3.1 Syntax

The grammar of *plain processes* is defined as follows:

$$P, Q, R := 0 \quad | \quad P + Q \quad | \quad P \mid Q \quad | \quad \nu k.P \quad | \quad !\, P \quad | \\ \mathsf{let}\ x = u\ \mathsf{in}\ P\ \mathsf{else}\ Q \quad | \quad \mathsf{in}(u, x).P \quad | \quad \mathsf{out}(u, v).P$$

where u, v are terms, and x is a variable of $\mathcal{X}$. Our calculus contains the nil process 0, parallel composition $P \mid Q$, choice $P + Q$, input $\mathsf{in}(u, x).P$, output $\mathsf{out}(u, v)$, replication $\nu k.P$ that typically models nonce or key generation, and unbounded replication $!\, P$. Note that our calculus also contains the assignment of variables $\mathsf{let}\ x = u\ \mathsf{in}\ P\ \mathsf{else}\ Q$. In many calculus, $\mathsf{let}\ x = u\ \mathsf{in}\ P$ is considered as syntactic sugar for $P\{^{u}/_{x}\}$. However, since we consider the computation time of messages during the execution of a process, the operation $\mathsf{let}\ x = u\ \mathsf{in}\ P$ is not syntactic sugar anymore. For example, the three following processes do not yield the same computation time even though they send out the same messages.

- $P_1 = \mathsf{let}\ x = \mathsf{senc}(a, k).\ \mathsf{in}\ \mathsf{out}(c, \mathsf{h}(n)).\mathsf{out}(c, \langle x, x\rangle)$
- $P_2 = \mathsf{out}(c, \mathsf{h}(n)).\mathsf{let}\ x = \mathsf{senc}(a, k)\ \mathsf{in}\ \mathsf{out}(c, \langle x, x\rangle)$
- $P_3 = \mathsf{out}(c, \mathsf{h}(n)).\mathsf{out}(c, \langle \mathsf{senc}(a, k), \mathsf{senc}(a, k)\rangle)$

P_1 first computes $\mathsf{senc}(a, k)$, and then outputs $\mathsf{h}(n)$ and $\langle \mathsf{senc}(a, k), \mathsf{senc}(a, k)\rangle$. P_2 is very similar but outputs $\mathsf{h}(n)$ before computing $\mathsf{senc}(a, k)$ meaning that the output of $\mathsf{h}(n)$ will occur faster in P_2 than in P_1, thus an attacker may observe the difference. Finally, P_3 computes $\mathsf{senc}(a, k)$ twice and therefore takes twice more time.

The operation $\mathsf{let}\ x = u\ \mathsf{in}\ P$ can also be used to change the default evaluation strategy of terms. As mentioned in the previous section, we assume that all primitives are computed using the call-by-value evaluation strategy with a lazy evaluation when a failure arises. For example, the eager evaluation of a message $\mathsf{senc}(\mathsf{sdec}(y, k), u)$ in the process $\mathsf{let}\ x = \mathsf{senc}(\mathsf{sdec}(y, k), u)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ can be modelled with the following process:

$$\mathsf{let}\ x_1 = \mathsf{sdec}(y, k)\ \mathsf{in}\ \mathsf{let}\ x = \mathsf{senc}(x_1, u)\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \mathsf{else}\ \mathsf{let}\ x_2 = u\ \mathsf{in}\ Q\ \mathsf{else}\ Q$$

In this process, even if the computation of $\mathsf{sdec}(y, k)$ fails (else branch), then u is still computed.

Note that the else branch in $\mathsf{let}\ x = u\ \mathsf{in}\ P\ \mathsf{else}\ Q$ is used in case u cannot be computed. For example, $\mathsf{let}\ x = \mathsf{sdec}(a, a)\ \mathsf{in}\ 0\ \mathsf{else}\ \mathsf{out}(c, ok)$ would output ok. At last, note that the traditional conditional branching (If-then-else) is not part of our calculus. We use instead the assignment of variables and the destructor symbol equals. The traditional process $\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$ is thus replaced by $\mathsf{let}\ x = \mathsf{equals}(u, v)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ where x does not appear in P nor Q.

The computation time of some operation obviously depends on the machine on which the computation is performed. For example, a server is much faster than a biometric passport. We defined *extended processes* to represent different physical *machines* that can be running during the execution of a protocol. For example, biometric passports are distinct physical machines that can be observed independently. In contrast, a server runs several threads which cannot be distinguished from an external observer.

The grammar for our extended processes is defined as follows:

$$A, B := [P, i, T] \quad | \quad !\,A \quad | \quad A \,||\, B$$

where P is a plain process, i is an integer, and T is a set of time functions. $[P, i, T]$ represents a machine with program P and computation time induced by T. The integer i represents the computation time used so far on that machine. Note that inside a machine $[P, i, T]$, there can be several processes running in parallel, *e.g.* $P_1 \mid \ldots \mid P_n$. We consider that their executions rely on a scheduling on a single computation machine and so the computation time might differ depending on the scheduling. The situation is different in the case of a real parallel execution of two machines, *e.g.* $A \,||\, B$ where the attacker can observe the execution of A and B independently.

Messages are made available to the attacker through *frames*. Formally, we assume a set of variables $\mathcal{AX}$, disjoint from $\mathcal{X}$. Variables of $\mathcal{AX}$ are typically denoted $ax_1, \ldots, ax_n$. A frame is an expression of the form $\Phi = \{ax_1 \rhd u_1; \ldots; ax_n \rhd u_n\}$ where $ax_i \in \mathcal{AX}$ and u_i are terms. The application of a frame Φ to a term M, denoted $M\Phi$, is defined as for the application of substitutions.

Definition 2 (time process). *A* time process *is a tuple* $(\mathcal{E}, A, \Phi, \sigma)$ *where:*

- $\mathcal{E}$ *is a set of names that represents the private names of* A*;*
- Φ *is a ground frame with domain included in* $\mathcal{AX}$*. It represents the messages available to the attacker;*
- A *is an extended process;*
- σ *is a substitution of variables to ground terms. It represents the current memory of the machines in* A*.*

3.2 Semantics

The semantics for time processes explicits the computation time of each operation. In particular, for each operation, we define a specific time function representing its computation time standalone, *i.e.* without considering the computation time required to generate the messages themselves. Hence, given a set T of time functions associated a physical machine, $\mathsf{t\_letin}_T(n)$ represents the computation time of the assignation

of a message of length n to a variable, whereas $\mathtt{t\_letelse}_T$ represents the computation time in the case the computation of the message fails; $\mathtt{t\_in}_T(n)$ (resp. $\mathtt{t\_out}_T(n)$) corresponds to the computation time of the input (resp. output) of a message of length n; and $\mathtt{t\_comm}_T(n)$ corresponds to the computation time of the transmission of the message of length n through internal communication. At last, $\mathtt{t\_restr}_T(n)$ represents the time needed to generate a fresh nonce of length n.

The semantics for time processes is similar to the semantics of the applied-pi calculus [4] and is given in Figure 1. For example, the label $\mathsf{out}(M, ax_n, j)$ means that some message has been sent on a channel corresponding to M after some time j (j is actually the total computation time until this send action). This message is stored in variable ax_n by the attacker. Internal communications within the same machine (or group of machines connected through a local network) cannot be observed by an attacker, therefore the computation time of the corresponding machine increases but the transition is silent (τ action). No external machines can communicate secretly since we assume the attacker can control and monitor all communications (he can at least observe the encrypted traffic). Lastly, note that the choice, replication and parallel composition operators do not have associated time functions.

The $\xrightarrow{w}$ relation is the reflexive and transitive closure of $\xrightarrow{\ell}$, where w is the concatenation of all actions. Moreover, $\overset{\mathsf{tr}}{\Rightarrow}$ is the relation $\xrightarrow{w}$ where tr are the words w without the non visible actions (τ). The set of traces of a time process $\mathcal{P}$ is the set of the possible sequences of actions together with the resulting frame.

$$\mathsf{trace}(\mathcal{P}) = \left\{ (\mathsf{tr}, \nu\mathcal{E}'.\Phi') \;\middle|\; \mathcal{P} \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{E}', A', \Phi', \sigma') \text{ for some } \mathcal{E}', A', \Phi', \sigma' \right\}$$

Example 5. Consider the signature $\mathcal{F}$, the set L of length functions of Example 3 and the set T of time functions of Example 4 and assume that for all $n \in \mathbb{N}$, $\mathtt{t\_out}_T(n) = n$. Let $a, b \in \mathcal{N}_\ell$ and $k \in \mathcal{N}_{\ell_{pk}}$ with $\ell, \ell_{pk} \in \mathbb{N}$. Consider the time process $\mathcal{P} = (\emptyset, [\mathsf{out}(c, \langle a, b\rangle), 0, T] \,||\, [\mathsf{out}(c, \mathsf{aenc}(a, k)), 0, T], \emptyset, \emptyset)$. Since we have $\mathsf{len}_L(\mathsf{aenc}(a, k)) = \ell + \ell_{pk}$, $\mathsf{len}_L(\langle a, b\rangle) = 2\ell + 1$, $\mathsf{ctime}_{L,T}(\mathsf{aenc}(a, b), \emptyset) = \ell \cdot \ell_{pk}^3 + 2$ and $\mathsf{ctime}_{L,T}(\langle a, b\rangle, \emptyset) = 3$, the set $\mathsf{trace}(A)$ is composed of four traces (s, Φ):

1. $s = \mathsf{out}(c, ax_1, \ell \cdot \ell_{pk}^3 + \ell + \ell_{pk} + 3)$ and $\Phi = \{ax_1 \rhd \mathsf{aenc}(a, k)\}$
2. $s = \mathsf{out}(c, ax_1, 2\ell + 5)$ and $\Phi = \{ax_1 \rhd \langle a, b\rangle\}$
3. $s = \mathsf{out}(c, ax_1, \ell \cdot \ell_{pk}^3 + \ell + \ell_{pk} + 3).\mathsf{out}(c, ax_2, 2\ell + 5)$ and $\Phi = \{ax_1 \rhd \mathsf{aenc}(a, k); ax_2 \rhd \langle a, b\rangle\}$
4. $s = \mathsf{out}(c, ax_1, 2\ell + 5).\mathsf{out}(c, ax_2, \ell \cdot \ell_{pk}^3 + \ell + \ell_{pk} + 3)$ and $\Phi = \{ax_1 \rhd \langle a, b\rangle; ax_2 \rhd \mathsf{aenc}(a, k)\}$

Note that since each computation time is local to each machine, the last argument of the out action is not necessarily increasing globally on the trace, as exemplified by the third trace.

3.3 Example: the PA protocol

We consider (a simplified version of) the Passive Authentication protocol (PA), presented in [7]. It is designed for transmitting a secret without revealing the identity of

$$(\mathcal{E}, [\mathsf{let}\ x = u\ \mathsf{in}\ P\ \mathsf{else}\ Q \mid R, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{\tau} \quad \text{(LET)}$$
$$(\mathcal{E}, [P \mid R, j, T] \,||\, A, \Phi, \sigma \cup \{^{u\sigma\downarrow}/_{x}\})$$
$$\text{if } \mathsf{Message}(u\sigma) \text{ with } j = i + \mathsf{ctime}_{L,T}(u, \sigma) + \mathsf{t\_letin}_T(\mathsf{len}_L(u\sigma\downarrow))$$

$$(\mathcal{E}, [\mathsf{let}\ x = u\ \mathsf{in}\ P\ \mathsf{else}\ Q \mid R, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{\tau} (\mathcal{E}, [Q \mid R, j, T] \,||\, A, \Phi, \sigma) \quad \text{(ELSE)}$$
$$\text{if } \neg\mathsf{Message}(u\sigma) \text{ with } j = i + \mathsf{ctime}_{L,T}(u, \sigma) + \mathsf{t\_letelse}_T$$

$$(\mathcal{E}, [\mathsf{out}(u, t).Q_1 \mid \mathsf{in}(v, x).Q_2 \mid R, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{\tau} \quad \text{(COMM)}$$
$$(\mathcal{E}, [Q_1 \mid Q_2 \mid R, j, T] \,||\, A, \Phi, \sigma \cup \{^{t\sigma\downarrow}/_{x}\})$$
$$\text{if } \mathsf{Message}(u\sigma), \mathsf{Message}(v\sigma), \mathsf{Message}(t\sigma) \text{ and } u\sigma\downarrow = v\sigma\downarrow \text{ with } j = i +$$
$$\mathsf{ctime}_{L,T}(u, \sigma) + \mathsf{ctime}_{L,T}(v, \sigma) + \mathsf{ctime}_{L,T}(t, \sigma) + \mathsf{t\_comm}_T(\mathsf{len}_L(t\sigma\downarrow))$$

$$(\mathcal{E}, [\mathsf{in}(u, x).Q \mid P, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{in(N,M)} (\mathcal{E}, [Q \mid P, j, T] \,||\, A, \Phi, \sigma \cup \{^{t}/_{x}\}) \quad \text{(IN)}$$
$$\text{if } M\Phi\downarrow = t, \mathit{fvars}(M, N) \subseteq \mathrm{dom}(\Phi), \mathit{fnames}(M, N) \cap \mathcal{E} = \emptyset,$$
$$N\Phi\downarrow = u\sigma\downarrow, \mathsf{Message}(M\Phi), \mathsf{Message}(N\Phi), \text{ and } \mathsf{Message}(u\sigma)$$
$$\text{with } j = i + \mathsf{ctime}_{L,T}(u, \sigma) + \mathsf{t\_in}_T(\mathsf{len}_L(t))$$

$$(\mathcal{E}, [\mathsf{out}(u, t).Q \mid P, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{out(M, ax_n, j)} \quad \text{(OUT)}$$
$$(\mathcal{E}, [Q \mid P, j, T] \,||\, A, \Phi \cup \{ax_n \rhd t\sigma\downarrow\}, \sigma)$$
$$\text{if } M\Phi\downarrow = u\sigma\downarrow, \mathsf{Message}(u\sigma), \mathit{fvars}(M) \subseteq \mathrm{dom}(\Phi), \mathit{fnames}(M) \cap \mathcal{E} = \emptyset,$$
$$\mathsf{Message}(M\Phi), \mathsf{Message}(t\sigma) \text{ and } ax_n \in \mathcal{AX}, n = |\Phi| + 1$$
$$\text{with } j = i + \mathsf{ctime}_{L,T}(t, \sigma) + \mathsf{ctime}_{L,T}(u, \sigma) + \mathsf{t\_out}_T(\mathsf{len}_L(t\sigma\downarrow))$$

$$(\mathcal{E}, [P_1 + P_2 \mid R, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{\tau} (\mathcal{E}, [P_1 \mid R, i, T] \,||\, A, \Phi, \sigma) \quad \text{(CHOICE-1)}$$

$$(\mathcal{E}, [P_1 + P_2 \mid R, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{\tau} (\mathcal{E}, [P_2 \mid R, i, T] \,||\, A, \Phi, \sigma) \quad \text{(CHOICE-2)}$$

$$(\mathcal{E}, [\nu k.P \mid R, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{\tau} (\mathcal{E} \cup \{k\}, [P \mid R, j, T] \,||\, A, \Phi, \sigma) \quad \text{(RESTR)}$$
$$\text{with } j = i + \mathsf{t\_restr}_T(\ell) \text{ and } k \in \mathcal{N}_\ell$$

$$(\mathcal{E}, [!\, P \mid R, i, T] \,||\, A, \Phi, \sigma) \xrightarrow{\tau} (\mathcal{E}, [!\, P \mid P\rho \mid R, i, T] \,||\, A, \Phi, \sigma) \quad \text{(REPL)}$$

$$(\mathcal{E}, !\, A \,||\, B, \Phi, \sigma) \xrightarrow{\tau} (\mathcal{E}, !\, A \,||\, A\rho \,||\, B, \Phi, \sigma) \quad \text{(M-REPL)}$$

where u, v, t are ground terms, x is a variable and ρ is used to rename variables in $\mathit{bvars}(P)$ and $\mathit{bvars}(A)$ (resp. names in $\mathit{bnames}(P)$ and $\mathit{bnames}(A)$) with fresh variables (resp. names).

Fig. 1. Semantics

the participants. In this protocol, an agent A wishes to engage in communication with an agent B that is willing to talk to A. However, A does not want to compromise her privacy by revealing her identity or the identity of B more broadly. The participants A and B proceed as follows:

$$\begin{aligned} A \to B \ &: \ \mathsf{aenc}(\langle N_a, \mathsf{pk}(sk_A)\rangle, \mathsf{pk}(sk_B)) \\ B \to A \ &: \ \mathsf{aenc}(\langle N_a, \langle N_b, \mathsf{pk}(sk_B)\rangle\rangle, \mathsf{pk}(sk_A)) \\ &\quad\ \mathit{else} \quad \mathsf{aenc}(N_b, \mathsf{pk}(sk_B)) \end{aligned}$$

A first sends to B a nonce N_a and her public key encrypted with the public key of B. If the message is of the expected form then B sends to A the nonce N_a, a freshly

generated nonce N_b and his public key, all of this being encrypted with the public key of A. If the message is not of the right form or if B is not willing to talk with A, then B sends out a "decoy" message $\mathsf{aenc}(N_b, \mathsf{pk}(sk_B))$. Intuitively, this message should look like B's other message from the point of view of an outsider. This is important since the protocol is supposed to protect the identity of the participants.

This protocol can be modeled in our process algebra as follows:

$$\begin{aligned} B(b,a) \stackrel{\mathsf{def}}{=} & \ \mathsf{in}(c,x).\mathsf{let}\ y = \mathsf{adec}(x, sk_b)\ \mathsf{in} \\ & \quad \mathsf{let}\ z = \mathsf{equals}(\mathsf{proj}_2(y), \mathsf{pk}(sk_a))\ \mathsf{in}\ \nu n_b.\mathsf{out}(c, \mathsf{aenc}(M, \mathsf{pk}(sk_a))).0 \\ & \quad \mathsf{else}\ \nu n_{error}.\mathsf{out}(c, \mathsf{aenc}(n_{error}, \mathsf{pk}(sk_a))).0 \\ & \mathsf{else}\ 0. \\ A(a,b) \stackrel{\mathsf{def}}{=} & \ \nu n_a.\mathsf{out}(c, \mathsf{aenc}(\langle n_a, \mathsf{pk}(sk_a)\rangle, \mathsf{pk}(sk_b))).\mathsf{in}(c,z).0 \end{aligned}$$

where $M = \langle \mathsf{proj}_1(y), \langle n_b, \mathsf{pk}(sk_b)\rangle\rangle$. The process $A(a,b)$ represents the role A played by agent a with b while the process $B(b,a)$ represents the role B played by agent b with a.

4 Time Equivalence

Privacy properties such as anonymity, unlinkability, or ballot privacy are often stated as *equivalence properties* [29,9]. Intuitively, Alice's identity remains private if an attacker cannot distinguish executions from Alice from executions from Bob. Equivalence properties are also useful to express strong secrecy [19], indistiguishability from an ideal system [5], or game-based properties [27]. Several definitions of equivalence have been proposed such as trace equivalence [4], observational equivalence [4], or diff-equivalence [18]. In this paper, we focus on trace equivalence that we adapt to account for length and computation times.

The ability of the attacker is now characterized by three parameters: the set of cryptographic primitives, their corresponding length functions, and their corresponding computation times (w.r.t. the attacker). Later in the paper, for decidability, we will show that we can restrict the attacker to a finite set of names. So we define a *length signature*, usually denoted $\mathcal{F}_\ell$, as a tuple of a symbol functions signature $\mathcal{F}$, a set of names $\mathsf{N} \subseteq \mathcal{N}$, and a set of length functions L, *i.e.* $\mathcal{F}_\ell = (\mathcal{F}, \mathsf{N}, L)$. Similarly, we denote a *time signature*, usually denoted $\mathcal{F}_{ti}$, as a pair containing a length signature $\mathcal{F}_\ell$ and a set of time functions T corresponding to the signature in $\mathcal{F}_\ell$, *i.e.* $\mathcal{F}_{ti} = (\mathcal{F}_\ell, T)$.

4.1 Time static equivalence

The notion of static equivalence has been extensively studied (see *e.g.*, [3]). It corresponds to the indistinguishability of sequences of messages from the point of view of the attacker. In the standard definition of static equivalence [3,14,28], the attacker can only perform cryptographic operations on messages. [24] introduces *length static equivalence*, that provides the attacker with the ability to measure the length of messages. Intuitively, two frames are in length static equivalence if an attacker cannot see any difference, even when applying arbitrary primitives and measuring the length of the

resulting messages. In this framework, we also provide the attacker with the capability to measure computation times. We therefore adapt the definition of static equivalence to account for both length and computation times.

Definition 3. *Let $\mathcal{F}_{ti} = (\mathcal{F}_\ell, T)$ be a time signature with $\mathcal{F}_\ell = (\mathcal{F}, \mathsf{N}, L)$. Let $\mathcal{E}, \mathcal{E}'$ two sets of names. Let Φ and Φ' two frames. We say that $\nu\mathcal{E}.\Phi$ and $\nu\mathcal{E}'.\Phi'$ are time statically equivalent w.r.t. $\mathcal{F}_{ti}$, written $\nu\mathcal{E}.\Phi \sim_{ti}^{\mathcal{F}_{ti}} \nu\mathcal{E}'.\Phi'$, when $\mathrm{dom}(\Phi) = \mathrm{dom}(\Phi')$, $fnames(\nu\mathcal{E}.\Phi, \nu\mathcal{E}'.\Phi') \cap (\mathcal{E}' \cup \mathcal{E}) = \emptyset$ and when for all $i, j \in \mathbb{N}$, for all $M, N \in \mathcal{T}(\mathcal{F}, \mathsf{N} \cup \mathcal{X})$ such that $fvars(M, N) \subseteq \mathrm{dom}(\Phi)$ and $fnames(M, N) \cap (\mathcal{E} \cup \mathcal{E}') = \emptyset$, we have:*

- $\mathsf{Message}(M\Phi)$ *if and only if* $\mathsf{Message}(M\Phi')$
- *if* $\mathsf{Message}(M\Phi)$ *and* $\mathsf{Message}(N\Phi)$ *then*
  1. $M\Phi{\downarrow} = N\Phi{\downarrow}$ *if and only* $M\Phi'{\downarrow} = N\Phi'{\downarrow}$*; and*
  2. $\mathsf{len}_L(M\Phi{\downarrow}) = i$ *if and only if* $\mathsf{len}_L(M\Phi'{\downarrow}) = i$*; and*
  3. $\mathsf{ctime}_{L,T}(M, \Phi) = j$ *iff* $\mathsf{ctime}_{L,T}(M, \Phi') = j$

Consider the length signature $\mathcal{F}_\ell$, we say that $\nu\mathcal{E}.\Phi$ and $\nu\mathcal{E}'.\Phi'$ are length statically equivalent w.r.t. $\mathcal{F}_\ell$, written $\nu\mathcal{E}.\Phi \sim_{\ell}^{\mathcal{F}_\ell} \nu\mathcal{E}'.\Phi'$, when $\nu\mathcal{E}.\Phi$ and $\nu\mathcal{E}'.\Phi'$ satisfy the same properties as above except Property 3.

4.2 Time trace equivalence

Time trace equivalence is a generalization of time static equivalence to the active case. It corresponds to the standard trace equivalence [4] except that the attacker can now observe the execution time of the processes. Intuitively, two extended processes $\mathcal{P}$ and $\mathcal{Q}$ are in time trace equivalence if any sequence of actions of $\mathcal{P}$ can be matched by the same sequence of actions in $\mathcal{Q}$ such that the resulting frames are time statically equivalent. It is important to note that the sequence of actions now reflects the computation time of each action. We also recall the definition of length trace equivalence introduced in [24], which accounts for the ability to measure the length but not the computation time. We denote by $=_{\not t\!t}$ the equality of sequences of labels, where the time parameters of outputs are ignored. Formally, we define $\ell_1 \dots \ell_p =_{\not t\!t} \ell'_1 \dots \ell'_q$ to hold when $p = q$ and

- for all N, M, $\ell_i = in(N, M)$ if and only if $\ell'_i = in(N, M)$; and
- for all $M, ax_n, \ell_i = out(M, ax_n, c)$ for some c if and only if $\ell'_i = out(M, ax_n, c')$ for some c'.

Definition 4. *Consider a time (resp. length) signature $\mathcal{F}$. Let $\mathcal{P}$ and $\mathcal{Q}$ be two closed time processes with $fnames(\mathcal{P}, \mathcal{Q}) \cap bnames(\mathcal{P}, \mathcal{Q}) = \emptyset$. $\mathcal{P} \sqsubseteq_{ti}^{\mathcal{F}} \mathcal{Q}$ (resp. $\mathcal{P} \sqsubseteq_{\ell}^{\mathcal{F}} \mathcal{Q}$) if for every $(\mathsf{tr}, \nu\mathcal{E}.\Phi) \in \mathsf{trace}(\mathcal{P})$, there exists $(\mathsf{tr}', \nu\mathcal{E}.\Phi') \in \mathsf{trace}(\mathcal{Q})$ such that $\nu\mathcal{E}.\Phi \sim_{ti}^{\mathcal{F}} \nu\mathcal{E}'.\Phi'$ and $\mathsf{tr} = \mathsf{tr}'$ (resp. $\nu\mathcal{E}.\Phi \sim_{\ell}^{\mathcal{F}} \nu\mathcal{E}'.\Phi'$ and $\mathsf{tr} =_{\not t\!t} \mathsf{tr}'$).*

Two closed time processes $\mathcal{P}$ and $\mathcal{Q}$ are time (resp. length) trace equivalent w.r.t. *$\mathcal{F}$, denoted by $\mathcal{P} \approx_{ti}^{\mathcal{F}} \mathcal{Q}$ (resp. $\mathcal{P} \approx_{\ell}^{\mathcal{F}} \mathcal{Q}$), if $\mathcal{P} \sqsubseteq_{ti}^{\mathcal{F}} \mathcal{Q}$ and $\mathcal{P} \sqsubseteq_{ti}^{\mathcal{F}} \mathcal{Q}$ (resp. $\mathcal{P} \sqsubseteq_{\ell}^{\mathcal{F}} \mathcal{Q}$ and $\mathcal{P} \sqsubseteq_{\ell}^{\mathcal{F}} \mathcal{Q}$).*

4.3 Timing attacks against PA

We consider again the PA protocol described in Section 3.3. This protocol should in particular ensure the anonymity of the sender A. The anonymity of A can be stated as an equivalence property: an attacker should not be able to distinguish whether b is willing to talk to a (represented by the process $B(b,a)$) or willing to talk to a' (represented by the process $B(b,a')$), provided that a, a' and b are honest participants. This can be modeled by the following equivalence:

$$(\mathcal{E}, [B(b,a'),0,T] \,|||\, [A(a',b),0,T], \Phi, \emptyset) \overset{?}{\approx}^{\mathcal{F}}_{ti} (\mathcal{E}, [B(b,a),0,T] \,|||\, [A(a,b),0,T], \Phi, \emptyset)$$

with $\mathcal{E} = \{sk_a, sk_{a'}, sk_b\}$, $\Phi = \{ax_1 \rhd \mathsf{pk}(sk_a); ax_2 \rhd \mathsf{pk}(sk_{a'}); ax_3 \rhd \mathsf{pk}(sk_b)\}$.

In the literature, the Private Authentication protocol was proved [23] to preserve A's anonymity when considering standard trace equivalence, *i.e.* without length and time. However, an attacker can easily break anonymity by measuring the length of the messages. Indeed, it is easy to notice that the length of the decoy message is smaller than the size of the regular message. Therefore, an attacker may simply initiate a session with B in the name of A:

$$C(A) \to B \ : \ \mathsf{aenc}(\langle N_c, \mathsf{pk}(sk_A)\rangle, \mathsf{pk}(sk_B))$$

If the message received in response from B is "long", the attacker learns that B is willing to talk with A. If the message is "small", the attacker learns that A is not one of B's friends.

This attack can be easily reflected in our formalism. Consider the sequence of labels $\mathsf{tr}(j) = \mathsf{in}(c, \mathsf{aenc}(\langle n_i, ax_1\rangle, ax_3)).\mathsf{out}(c, ax_4, j)$ and the corresponding execution on $B(b,a)$, where b is indeed willing to talk with a.

$$(\mathcal{E}, [B(b,a),0,T] \,|||\, [A(a,b),0,T], \Phi, \emptyset) \overset{\mathsf{tr}(j)}{\Rightarrow} (\mathcal{E}', [A(a,b),0,T], \Phi \cup \{ax_4 \rhd M\}, \sigma)$$

with $M = \mathsf{aenc}(\langle n_i, \langle n_b, \mathsf{pk}(sk_b)\rangle\rangle, \mathsf{pk}(sk_a))$ and $\mathcal{E}' = \mathcal{E} \cup \{n_b\}$ for some σ and j. On the other hand, when the communication is between a' and b then b detects that the public key does not correspond to a' and outputs the decoy message:

$$(\mathcal{E}, [B(b,a'),0,T] \,|||\, [A(a',b),0,T], \Phi, \emptyset) \overset{\mathsf{tr}(j')}{\Rightarrow} (\mathcal{E}', [A(a,b),0,T], \Phi \cup \{ax_4 \rhd M'\}, \sigma')$$

with $M' = \mathsf{aenc}(n_{error}, \mathsf{pk}(sk_a))$ for some σ' and j'. If the attacker computes the length of the received message, he gets $\mathsf{len}_L(\mathsf{aenc}(\langle n_i, \langle n_b, \mathsf{pk}(sk_b)\rangle\rangle, \mathsf{pk}(sk_a))) = 2\ell + \ell_{pk} + 2$ and $\mathsf{len}_L(\mathsf{aenc}(n_{error}, \mathsf{pk}(sk_a))) = \ell$ with $n_i, n_b, n_{error} \in \mathcal{N}_\ell$ and $sk_b \in \mathcal{N}_{pk}$. Therefore the two resulting frames are not in length static equivalence, thus

$$(\mathcal{E}, [B(b,a'),0,T] \,|||\, [A(a',b),0,T], \Phi, \emptyset) \not\approx^{\mathcal{F}}_{ti} (\mathcal{E}, [B(b,a),0,T] \,|||\, [A(a,b),0,T], \Phi, \emptyset)$$

To repair the anonymity of the PA protocol, the decoy message should have the same length than the regular message.

PA-fix1 A first solution is to include N_a in the decoy message which is set to be $\mathsf{aenc}(\langle N_a, \mathsf{Error}\rangle, \mathsf{pk}(sk_A))$ where Error is a constant of same length than $\langle N_b, \mathsf{pk}(sk_B)\rangle$. However, this variant does not satisfy even trace equivalence since the attacker can now reconstruct $\mathsf{aenc}(\langle N_a, \mathsf{Error}\rangle, \mathsf{pk}(sk_A))$ when N_a has been forged by himself.

PA-fix2 To fix this attack, a natural variant is to set the decoy message to be $\mathsf{aenc}(\langle N_a, N_d\rangle, \mathsf{pk}(sk_A))$, where N_d is a nonce of same length than $\langle N_b, \mathsf{pk}(sk_B)\rangle$. However, this variant is now subject to a timing attack. Indeed, it takes more time to generate N_d than N_b since N_d is larger. Therefore an attacker may still notice the difference. Note that this attack cannot be detected when considering length trace equivalence only.

PA-fix3 Finally, a third solution is to set the decoy message to be the cipher $\mathsf{aenc}(\langle N_a, \langle N_b, \mathsf{Error}\rangle, \mathsf{pk}(sk_A)\rangle)$ where Error is a constant of same length than $\mathsf{pk}(sk_B)$. We show in Section 6 that due to our main result and thanks to the APTE tool, we are able to prove this version secure, assuming that public keys are of the same length (otherwise there is again a straightforward attack on privacy).

We will see in Section 6 that our tool detects all these attacks.

5 Reduction of time trace equivalence to length equivalence

We focus in this section on the key result of this paper: time equivalence reduces to length equivalence. We show that this holds for arbitrary processes, possibly with replications and private channels (Theorem 1). This means that, from a decidability point of view, there is no need to enrich the model with time. We also prove that our result induces that time trace equivalence for processes without replication can also be reduced to length trace equivalence for processes without replication, even if we restrict the names of the attacker. Finally, applying the decidability result on length trace equivalence of [24], we can deduce decidability of trace equivalence for processes without replication and for a fixed signature that includes all standard cryptographic primitives (Theorem 2).

These three results rely on a generic transformation from a time process P to a process P' where the sole observation of the length of the messages exchanged in P' reflects both the time and length information leaked by P.

5.1 Representing computation time with messages

The key idea to get rid of computation times is to attach to each term t a special message, called *time message*, whose length corresponds to the time needed to compute t. To that extent, we first need to augment the signature used to describe our processes. Given a time signature $\mathcal{F}_t = ((\mathcal{F}, \mathsf{N}, L), T)$, we extend it as $\overline{\mathcal{F}_t}^T = ((\overline{\mathcal{F}}^T, \mathsf{N}, \overline{L}^T), \overline{T}^T)$, which is defined as follows. We first add, for each function symbol f, a fresh function symbol $\bar{\mathsf{f}}$ whose length function is the time function of f, meaning that $\mathsf{len}^{\bar{\mathsf{f}}}_{\overline{L}^T} = \mathsf{time}^{\mathsf{f}}_T$. Similarly, for each action proc in the execution of a process, we add a new function symbol whose length function represents the computation time of proc, that is $\mathsf{len}^{\mathsf{g}_{\mathsf{proc}}}_{\overline{L}^T} = \mathsf{t\_proc}_T$. Lastly, we consider two new symbol functions $\mathsf{plus}/1$ and $\mathsf{hide}/2$ where the resulting size of the application of plus is the sum of the size of its arguments, and hide reveals only the size of its first argument. Since these news function symbols should not yield information on the computation time other than by their size, we consider that all their time functions are the null function. With these extended time signature $\overline{\mathcal{F}_t}^T$, the time

message of a term t, denoted $[t]_{L,T}$, can be naturally defined. For instance, if $t = \mathsf{f}(t_1, \ldots, t_m)$ then $[t]_{L,T} = \mathsf{plus}([t_1]_{L,T}, \ldots \mathsf{plus}([t_m]_{L,T}, \bar{\mathsf{f}}(t_1, \ldots, t_m)) \ldots)$. Thanks to the function symbol plus, the length of $[t]_{L,T}$ models exactly the computation time of t.

5.2 Transformed processes

The computation time of a process becomes visible to an attacker only at some specific steps of the execution, typically when the process sends out a message. Therefore the corresponding time message should consider all previous actions since the last output. In case a machine executes only a sequential process (*i.e.* that does not include processes in parallel) then the computation time of internal actions is easy to compute. For example, given a process $P = \mathsf{in}(c, x).\nu k.\mathsf{out}(c, v)$, the computation time of P when v is output can be encoded using the following time message $\mathsf{plus}(m_{in}, \mathsf{plus}(\mathsf{g}_{\mathsf{restr}}(k), m_{out}))$ where:

$$m_{in} = \mathsf{plus}([x]_{L,T}, \mathsf{plus}([c]_{L,T}, \mathsf{g}_{\mathsf{in}}(x))) \quad m_{out} = \mathsf{plus}([v]_{L,T}, \mathsf{plus}([c]_{L,T}, \mathsf{g}_{\mathsf{out}}(v)))$$

However, if a machine executes a process Q in parallel of P, then the time message m does not correspond anymore to the computation time when v is output since some actions of Q may have been executed between the actions of P. Therefore, we need to "store" the computation time that has elapsed so far. To do this, we introduce *cells* that can store the time messages of a machine and will be used as time accumulator. Formally, a cell is simply a process with a dedicated private channel defined as $Cell(c, u) = \mathsf{out}(c, u) \mid !\, \mathsf{in}(c, x).\mathsf{out}(c, x)$. Note that a cell can only alternate between inputs and outputs (no consecutive outputs can be done). Thanks to those cells, we can define a transformation for a time process P into an equivalent process w.r.t. to some cell d and some length and time functions L and T respectively, denoted $[P]^d_{L,T}$, where the computation time can now be ignored.

Intuitively, each action of a plain process first starts by reading in the cell d and always ends by writing on the cell the new value of the computation time. For instance, $[\nu k.P]^d_{L,T} = \mathsf{in}(d, y).\nu k.\mathsf{out}(d, \mathsf{plus}(y, \mathsf{g}_{\mathsf{restr}}(k))).[P]^d_{L,T}$. Moreover, in the case of an output, $\mathsf{out}(u, v)$ is transformed to $\mathsf{out}(u, \langle v, \mathsf{hide}(t, k)\rangle)$ where t is the current value of the computation time of the plain process and k is a fresh nonce. Hence, the attacker gets information about the computation time of the process through the size of the second message of the output. The most technical case is for the process $\mathsf{let}\ x = u\ \mathsf{in}\ P\ \mathsf{else}\ Q$. Indeed, if u is not a message then the process executes Q instead of P. The main issue here is that the computation time of u depends on which subterm makes the computation fail. This, in turn, may depend on the intruder's inputs. Therefore we introduce below the process $\mathsf{LetTr}_T(c, t, [u], y)$ that determines which cryptographic primitive fails and then returns on channel c the computation time message that corresponds to the execution of u, added to the existing computation time message y and t being some initial parameters.

$\mathsf{LetTr}_T(c,t,\emptyset,u) = \mathsf{out}(c,\mathsf{plus}(u,t))$
$\mathsf{LetTr}_T(c,t,[t_1;\ldots;t_n],u) = \mathsf{LetTr}_T(c,t,[t_2;\ldots;t_n],\mathsf{plus}(u,[t_1]_{L,T}))$ if $t_1 \in \mathcal{N} \cup \mathcal{X}$
$\mathsf{LetTr}_T(c,t,[t_1;\ldots;t_n],u) = \mathsf{let}\ x = t_1\ \mathsf{in}$
$\quad \mathsf{LetTr}_T(c,t,[t_2;\ldots;t_n],\mathsf{plus}(u,[t_1]_{L,T}))\ \mathsf{else}\ \mathsf{LetTr}_T(c,t',[v_1;\ldots;v_m],u)$
$\qquad\qquad$ where $t_1 = \mathsf{f}(v_1,\ldots,v_m)$, $t' = \overline{\mathsf{f}}(v_1,\ldots,v_m)$.

Thanks to this process, the transformed process $[\mathsf{let}\ x = u\ \mathsf{in}\ P\ \mathsf{else}\ Q]^d_{L,T}$ is defined as follows where $u = \mathsf{f}(v_1,\ldots,v_m)$, $t = \overline{\mathsf{f}}(v_1,\ldots,v_m)$.

$\mathsf{in}(d,y).\mathsf{let}\ x = u\ \mathsf{in}\ \mathsf{out}(d,\mathsf{plus}(\mathsf{plus}(y,\mathsf{g}_{\mathsf{letin}}(x)),[u]_{L,T})).[P]^d_{L,T}$
$\mathsf{else}\ \nu c.\ \big(\mathsf{LetTr}_T(c,t,[v_1;\ldots;v_m],\mathsf{plus}(y,\mathsf{g}_{\mathsf{letelse}})) \mid \mathsf{in}(c,z).\mathsf{out}(d,z).[Q]^d_{L,T}\big)$

This transformation is naturally adapted to extended processes by introducing a cell for each extended process $A = [P,i,T]$, that is $[A]_L = [\nu d.(Cell(d,n^i) \mid [P]^d_{L,T}),i,T]$ for some $n^i \in \mathcal{N}$.

5.3 Main theorem

We can finally state the main results of this paper. First, time equivalence can be reduced to length equivalence, for any two processes.

Theorem 1. *Let $\mathcal{F}_{ti} = ((\mathcal{F},\mathcal{N},L),T)$ be a time signature. Intuitively, T is the set of time functions for the attacker. Consider two time processes $\mathcal{P}_1 = (\mathcal{E}_1,A_1,\Phi_1,\emptyset)$ and $\mathcal{P}_2 = (\mathcal{E}_2,A_2,\Phi_2,\emptyset)$ with $\mathrm{dom}(\Phi_2) = \mathrm{dom}(\Phi_1)$, built on $(\mathcal{F},\mathcal{N},L)$ and time functions sets $T_1,\ldots,T_n$. Let $\mathcal{P}'_1 = (\mathcal{E}_1,[A_1]_L,\Phi_1,\emptyset)$ and $\mathcal{P}'_2 = (\mathcal{E}_2,[A_2]_L,\Phi_2,\emptyset)$. Then*

$$\mathcal{P}_1 \approx^{\mathcal{F}_{ti}}_{ti} \mathcal{P}_2 \text{ if, and only if, } \mathcal{P}'_1 \approx^{\overline{\mathcal{F}_{ti}}^{T,T_1,\ldots,T_n}}_{\ell} \mathcal{P}'_2$$

This theorem holds for arbitrary processes and for any signature and associated rewriting system. It is interesting to note that it also holds for arbitrary time functions. Moreover, the transformed processes $\mathcal{P}'_1$ and $\mathcal{P}'_2$ only add length functions which are either linear or are the same than the initial time functions. It therefore does not add any complexity. Note also that if $\mathcal{P}_1$ and $\mathcal{P}_2$ are two processes without replication then $\mathcal{P}'_1$ and $\mathcal{P}'_2$ are still processes with replication. For decidability in the case of processes without replication, we need to further restrict the number of names given to the attacker. We therefore refine our theorem for processes without replication with a slightly different transformation. Instead of adding cells of the form $\mathsf{out}(c,u) \mid !\,\mathsf{in}(c,x).\mathsf{out}(c,x)$, we unfold in advance the replication as much as needed in the extended process. As a consequence, and relying on the decidability of time trace equivalence described in [24], we can immediately deduce decidability of time trace equivalence for processes without replication and polynomial time functions.

Theorem 2. *Let $\mathcal{F}_{ti} = ((\mathcal{F},\mathcal{N},L),T)$ be a time signature such that $\mathcal{F} = \mathcal{F}_{\mathsf{stand}} \uplus \mathcal{F}_o$ where $\mathcal{F}_o$ contains only one-way symbols, that are not involved in any rewrite rules. We assume that L and T contain only polynomial functions. Then time trace equivalence is decidable for time processes without replication.*

Anonymity	Status	Execution time
PA-Original	timing attack	0.01 sec
PA-fix1	timing attack	0.01 sec
PA-fix2	timing attack	0.08 sec
PA-fix3	safe	0.3 sec

Unlinkability	Status	Execution time
BAC	timing attack	0.08 sec
AKA	timing attack	0.9 sec

Fig. 2. Timing attacks found with the APTE tool.

6 Application to privacy protocols

The APTE tool [21,22] is a tool dedicated to the automatic proof of trace equivalence of processes without replication, for the standard cryptographic primitives. It has been recently extended to length trace equivalence [24]. We have implemented our generic transformation (Theorem 2) and thanks to this translator from time to length equivalence, APTE can now be used to check time trace equivalence. Using the tool, we have studied the privacy of three protocols:

PA Our running example is the Private Authentication Protocol, described in Section 3.3. As explained in Section 4.3, this protocol suffers from length or time attacks for several versions of it, depending on the decoy message. With the APTE tool, we have found privacy attacks against all the fixes we first proposed. The APTE tool was able to show privacy of our last version of PA.

BAC As explained in the Introduction, several protocols are embedded in biometric passports, to protects users' privacy. We have studied the *Basic Access Control protocol* (BAC). With the APTE tool, we have retrieved the timing attack reported in [25]. Note that this attack could not have been detected when considering length trace equivalence only. Indeed, the returned message does not vary. The only noticeable change is the time needed to reply. Even if APTE is guaranteed to always terminate (since it implements a decidable procedure [21]), the corrected version that includes a fake test was unfortunately out of reach of the APTE tool in its current version (we stopped the computation after two days). This is due to the fact that the BAC protocol contains several inputs and else branches which causes state-explosion in APTE.

3G AKA protocol The 3G AKA protocol is deployed in mobile telephony to protect users from being traced by third parties. To achieve privacy, it makes use of temporary pseudonyms but this was shown to be insufficient [10]. Indeed, thanks to error messages, an attacker may recognize a user by replaying an old session. The

suggested fix proposes to simply use a unique error message. However, the protocol then remains subject to potential timing attacks (as for the BAC protocol). The APTE tool is able to automatically detect this timing privacy attack.

Our study is summarized in Figure 2. The precise specification of the protocols and their variants can be found in [20].

References

1. http://nacl.cr.yp.to/.
2. Machine readable travel document. Technical Report 9303, International Civil Aviation Organization, 2008.
3. M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, 2006.
4. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM Symp. on Principles of Programming Languages (POPL'01)*, 2001.
5. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *4th Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997.
6. Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, January 2005.
7. Martín Abadi and Cédric Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.
8. Jos Bacelar Almeida, Manuel Barbosa, Gilles Barthe, and François Dupressoir. Certified computer-aided cryptography: Efficient provably secure machine code from high-level implementations. In *21st ACM Conference on Computer and Communications Security (CCS'13)*, 2013.
9. M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *23rd IEEE Computer Security Foundations Symposium (CSF'10)*, 2010.
10. Myrto Arapinis, Loretta Ilaria Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Redon, and Ravishankar Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *ACM Conference on Computer and Communications Security*, pages 205–216, 2012.
11. Michael Backes, Goran Doychev, and Boris Köpf. Preventing side-channel leaks in web traffic: A formal approach. In *Network and Distributed System Security Symposium (NDSS'13)*, 2013.
12. Michael Backes, Markus Duermuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic emanations of printers. In *19th USENIX Security Symposium*, 2010.
13. Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic discovery and quantification of information leaks. In *Symposium on Security and Privacy (S&P'09)*, 2009.
14. Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. *ACM Transactions on Computational Logic*, 14, 2013.
15. G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In *5th European Symposium on Research in Computer Security (Esorics'98)*, volume 1485 of *LNCS*. Springer, 1998.
16. Daniel J. Bernstein, Tung Chou, and Peter Schwabe. Mcbits: Fast constant-time code-based cryptography. In *Cryptographic Hardware and Embedded Systems (CHES 2013)*, volume 8086 of *LNCS*, pages 250–272. Springer, 2013.

17. Fabrizio Biondi, Axel Legay, Pasquale Malacaria, , and Andrzej Wasowski. Quantifying information leakage of randomized protocols. In *14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'13)*, 2013.
18. B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th Symposium on Logic in Computer Science (LICS'05*, 2005.
19. Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *Symposium on Security and Privacy (S&P'04)*, pages 86–100. IEEE Comp. Soc. Press, 2004.
20. V. Cheval. APTE (Algorithm for Proving Trace Equivalence), 2013. `http://projects.lsv.ens-cachan.fr/APTE/`.
21. V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *18th ACM Conference on Computer and Communications Security (CCS'11)*, 2011.
22. Vincent Cheval. Apte: an algorithm for proving trace equivalence. In Erika Ábrahám and JKlaus Havelund, editors, *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, LNCS, Grenoble, France, April 2014. Springer. to appear.
23. Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. In *2nd International Conference on Principles of Security and Trust (POST'13)*, LNCS, pages 226–246. Springer, 2013.
24. Vincent Cheval, Véronique Cortier, and Antoine Plet. Lengths may break privacy – or how to check for equivalences with length. In *25th International Conference on Computer Aided Verification (CAV'13)*, volume 8043 of *LNCS*, pages 708–723. Springer, 2013.
25. Tom Chothia and Vitaliy Smirnov. A traceability attack against e-passports. In *14th International Conference on Financial Cryptography and Data Security*, 2010.
26. E. Cohen. Taps: A first-order verifier for cryptographic protocols. In *13th IEEE Computer Security Foundations Workshop (CSFW00)*. IEEE Computer Society, 2000.
27. H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *15th Conf. on Computer and Communications Security (CCS'08)*, 2008.
28. Véronique Cortier and Stéphanie Delaune. Decidability and combination results for two notions of knowledge in security protocols. *Journal of Automated Reasoning*, 48, 2012.
29. Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, (4):435–487, July 2008.
30. N. Evans and S. Schneider. Analysing time dependent security properties in CSP using PVS. In *6th European Symposium on Research in Computer Security (Esorics'00)*, 2000.
31. R. Gorrieri, E. Locatelli, and F. Martinelli. A simple language for real-time cryptographic protocol analysis. In *12th Eur. Symposium on Programming (ESOP'03)*, page 114128, 2003.
32. Gizela Jakubowska and Wojciech Penczek. Modelling and checking timed authentication of security protocols. *Fundamenta Informaticae*, pages 363–378, 2007.
33. Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant aes-gcm. In *Cryptographic Hardware and Embedded Systems (CHES 2009)*, volume 5747 of *LNCS*, pages 1–17. Springer, 2009.
34. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *16th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '96)*, pages 104–113. Springer-Verlag, 1996.
35. Boris Köpf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *14th ACM Conf. on Computer and Communications Security (CCS'07)*, 2007.
36. David Molnar, Matt Piotrowski, David Schultz, and David Wagner. The program counter security model: Automatic detection and removal of control-flow side channel attacks. In *Int. Conference and Information Security and Cryptology (ICISC'05)*, pages 156–168, 2005.
37. Quoc-Sang Phan, Pasquale Malacaria, Oksana Tkachuk, and Corina S. Pasareanu. Symbolic quantitative information flow. In *ACM SIGSOFT Software Engineering Notes*, 2012.