Tests for establishing security properties

Vincent Cheval[1,2], Stéphanie Delaune[3], and Mark Ryan[1]

[1] LORIA, CNRS & Inria Nancy - Grand Est
[2] School of Computer Science, University of Birmingham
[3] LSV, CNRS & ENS Cachan

Abstract. Ensuring strong security properties in some cases requires participants to carry out tests during the execution of a protocol. A classical example is electronic voting: participants are required to verify the presence of their ballots on a bulletin board, and to verify the computation of the election outcome. The notion of *certificate transparency* is another example, in which participants in the protocol are required to perform tests to verify the integrity of a certificate log.
We present a framework for modelling systems with such 'testable properties', using the applied pi calculus. We model the tests that are made by participants in order to obtain the security properties. Underlying our work is an attacker model called "malicious but cautious", which lies in between the Dolev-Yao model and the "honest but curious" model. The malicious-but-cautious model is appropriate for cloud computing providers that are potentially malicious but are assumed to be cautious about launching attacks that might cause user tests to fail.

1 Introduction

Security protocols are short distributed programs designed to achieve a security goal, such as authentication or secure messaging. In an ideal situation, the security goal is guaranteed to participants provided they adhere to the protocol. Sometimes, however, it is not possible to devise a protocol which is certain to achieve the goal; instead, the protocol guarantees that, if the goal fails, then the participants have a means to discover that fact.

Electronic voting protocols are such an example. In voting protocols, the participants are voters, and it is assumed that the election authorities and the network and other infrastructure is controlled by the attacker. Under those assumptions, there is no protocol which guarantees that the authorities declare the correct election outcome. Therefore, protocols aim for a weaker property, namely, if the election authorities declare an incorrect outcome then the participants can discover the fact. More precisely, the protocol outputs certain data, and the participants have a set of tests they perform on the data, and if the tests return a positive result then the declared outcome is guaranteed to be correct [11], or correct with high probability [5]. One can apply this idea to the incoercibility property of voting systems: in *Caveat Coercitor*, voters can perform a test to determine if certain kinds of coercion have taken place [10]. Bitcoin [14] is also an example: a Bitcoin user being offered a bitcoin can be sure that it has not already been spent only by performing tests on the public log of spent bitcoins.

In this paper, we propose a model and a framework to analyse protocols where partipants run tests on output data in order to determine if the expected properties hold. We call such properties *testable properties*. We elaborate two case studies. The first one is about *certificate transparency*, which is a protocol designed to provide greater evidence about the trustworthiness of public-key certificates [12]. Certificate transparency relies on certificate authorities maintaining a public log, and on client browsers performing tests on the log. The desired security property is expected to hold if the tests are positive. The second case study is about the French electronic passport, which has been shown not to satisfy expected privacy properties in some circumstances [3]. Although the French passport user can't guarantee that her privacy is preserved, she can perform a test to detect if her privacy may have been violated. If the test is negative, then the desired privacy property holds.

Instead of thinking that the protocols we consider provide testable properties (that is, properties that hold conditionally on certain run-time tests), one may consider that the original properties hold, but in the context of a weaker attacker that is not willing to take actions that can be detected by the tests. This view corresponds to situations in which the "attacker" is actually a service provider; in voting, it is the election authority, and in certificate transparency, it is the network provider. Such service providers have an incentive to give good service, and therefore traditional Dolev-Yao attacker model is too strong for them. To cope with this issue, an alternative model of an attacker has been considered in the literature: *honest-but-curious*. Roughly, an honest-but-curious provider will follow the protocol but he will also try to derive some information from the messages he learned during the execution. However, this model is too weak for our examples; there is simply no reason to suppose that the service provider will not engage in active attacks if he can do so undetectably. We adopt the term "malicious-but-cautious"; the service provider is assumed to be malicious if he can get away with it, but cautious in not leaving any verifiable evidence of its misbehaviour. This attacker model is related to the *covert adversary* [1] and *active security adversary* [8] introduced in the setting of secure multiparty computation. Malicious-but-cautious attackers are weaker than the all-powerful Dolev-Yao attackers, but stronger than the honest-but-curious attackers that confine themselves to passive attacks.

Our contributions. We develop a model and framework based on the applied pi calculus to analyse security protocols that rely on participants carrying out tests. Our model includes new primitives to allow participants to record information which will later be inputs to the tests. We specify a language for formulating the tests that are performed by participants. We illustrate our framework with two case studies. One of the case studies, called certificate transparency, is a new protocol which has not yet been studied by the academic community. We give a model and formalisation for the first time.

2 Model for security protocols

In this section, we introduce the cryptographic process calculus that we will use for describing protocols. This calculus is close to the applied pi calculus as defined in [2]. However, in order to model tests performed on logs after an execution, we add a special construct (namely rec) to allow participants of a protocol to record the information that will be used for that purpose.

2.1 Messages

A protocol consists of some agents communicating on a network. The messages sent by the agents are modelled using an abstract term algebra. For this, we assume an infinite set $\mathcal{N}$ of *names* which are used for representing keys, nonces, channels, and also public data like agent names. We also consider an infinite set $\mathcal{X}$ of *variables*, and a *signature* $\mathcal{F}$ consisting of a finite set of function symbols.

Terms are defined as names, variables, and function symbols applied to other terms. Let $N \subseteq \mathcal{N}$, $X \subseteq \mathcal{X}$ and $F \subseteq \mathcal{F}$. The set of terms built from N and X by applying function symbols from F is denoted $\mathcal{T}(F, X \cup N)$. We write $fv(u)$ (resp. $fn(u)$) for the set of variables (resp. names) occurring in a term u. A term is *closed* if it does not contain any variable.

To model algebraic properties of cryptographic primitives, we define an *equational theory* by a finite set E of equations $u = v$ with $u, v \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, *i.e., u* and v do not contain names. We define $=_{\mathsf{E}}$ to be the smallest equivalence relation on terms, that contains E and that is closed under application of function symbols and substitutions of terms for variables.

Example 1. A typical signature that can be used to model security protocols that rely on signature, asymmetric encryption, and list is $\mathcal{F}_{\mathsf{sign}}$ where:

$\mathcal{F}_{\mathsf{sign}} = \{\mathsf{sign}, \mathsf{getmsg}, \mathsf{vk}, \mathsf{aenc}, \mathsf{adec}, \mathsf{pk}, \langle\ \rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{h}, ::, \mathsf{head}, \mathsf{tail}, \bot\}$.

The function symbols sign, getmsg, and vk are used to represent signatures, whereas aenc, adec, and pk are used to model asymmetric encryption. We consider hashes (of arity 2), pairs and operators to manipulate lists. Then, we consider the equational theory $\mathsf{E}_{\mathsf{sign}}$, defined by the following equations ($i \in \{1, 2\}$):

$$\begin{array}{rrr}
\mathsf{getmsg}(\mathsf{sign}(x,y),\mathsf{vk}(y)) = x & \mathsf{proj}_1(\langle x_1, x_2\rangle) = x_1 & \mathsf{head}(x::y) = x \\
\mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) = x & \mathsf{proj}_2(\langle x_1, x_2\rangle) = x_2 & \mathsf{tail}(x::y) = y
\end{array}$$

Many interesting security properties, in particular privacy-type properties (*e.g.* in [3,4,9]) are formalised relying on the notion of *static equivalence* that compares frames. A frame is a sequence of the form $\nu\,\mathcal{E}.[w_1 \rhd u_1, \ldots, w_n \rhd u_n]$ where $\mathcal{E}$ is a finite set of restricted names (intuively the fresh ones), and the remaining part can be seen as a substitution with domain $\mathrm{dom}(\varphi) = \{w_1, \ldots, w_n\}$. The variables w_i enable us to refer to ground terms u_i. We denote by $=_\alpha$ the relation between frames that corresponds to α-renaming of bound names. Two frames are equivalent when the attacker cannot detect the difference between the two situations they represent, that is, his ability to distinguish whether two recipes M and N produce the same term does not depend on the frame.

Definition 1. *Let φ be a frame, and $M, N \in \mathcal{T}(\mathcal{F}, \mathcal{N} \cup \mathcal{X})$. We say that M and N are* equal in *the frame φ, and write $(M = N)\varphi$, if there exists $\mathcal{E}$ such that $\varphi =_\alpha \mathsf{new}\ \mathcal{E}.\sigma$, $(fn(M) \cup fn(N)) \cap \mathcal{E} = \emptyset$, and $M\sigma =_\mathsf{E} N\sigma$.*

We say that two frames φ and φ' are statically equivalent*, and write $\varphi \sim \varphi'$ when $\mathrm{dom}(\varphi) = \mathrm{dom}(\varphi')$; and for all $M, N \in \mathcal{T}(\mathcal{F}, \mathcal{N} \cup \mathcal{X})$, we have that:*

$$(M = N)\varphi \Leftrightarrow (M = N)\varphi'.$$

Example 2. Relying on the signature and equational theory introduced in Example 1, let $\varphi_v = \mathsf{new}\{sk\}.[w_1 \rhd \mathsf{aenc}(v, \mathsf{pk}(sk))]$. We have that $\varphi_{yes} \sim \varphi_{no}$. Intuitively, the equivalence holds since the attacker is not able to reconstruct the ciphertext or to open it. However, we have that $\varphi'_{yes} \not\sim \varphi'_{no}$ where:

$$\varphi'_v = \mathsf{new}\{sk\}.[w_1 \rhd \mathsf{aenc}(v, \mathsf{pk}(sk)),\ w_2 \rhd \mathsf{pk}(sk)].$$

2.2 Processes

Our *processes* are as in the applied pi calculus [2], except for the record message $\mathsf{rec}(x, v).P$ construct discussed below. Moreover, the applied pi calculus relies extensively on the renaming of bound variables and names. This is practical in presence of replication (! construct) but becomes a problem when one needs to refer to some particular variable and/or name after a protocol execution. In order to have a reliable way to talk about variables and names, we decorate each bang operator with an index, and we introduce the notion of *pattern.*

$$\begin{array}{rll} \overline{u} := & x[\mathsf{i}_1, \ldots, \mathsf{i}_k] & \text{variable pattern} \\ \mid & n[\mathsf{i}_1, \ldots, \mathsf{i}_k] & \text{name pattern} \\ \mid & \mathsf{f}(\overline{u_1}, \ldots, \overline{u_n}) & \text{function symbol application} \end{array}$$

where each i_j is either an index variable or an integer. The index variables will be instantiated each time the bang operator carrying this index will be unfolded. Using this notation, the set $\mathcal{N}$ of names (resp. $\mathcal{X}$ of variables) introduced in the previous section is made up of element of the form $n[i_1, \ldots, i_k]$ (resp. $x[i_1, \ldots, i_k]$) with $i_1, \ldots, i_k \in \mathbb{N}$. Note that a term is a pattern. We sometimes write $\overline{n}$ (resp. $\overline{x}$) instead of $n[\mathsf{i}_1, \ldots, \mathsf{i}_k]$ (resp. $x[\mathsf{i}_1, \ldots, \mathsf{i}_k]$). We also write n instead of $n[]$.

The grammar of our *processes* is as follows:

$$\begin{array}{rl} P, Q := & 0 \mid (P \mid Q) \mid \mathsf{new}\,\overline{n}.P \mid !^{\mathsf{i} \geq j}\, P \mid \mathsf{if}\ \overline{u} = \overline{v}\ \mathsf{then}\ P\ \mathsf{else}\ Q \\ & \mid \mathsf{in}(\overline{u}, \overline{x}).P \mid \mathsf{out}(\overline{u}, \overline{v}).P \mid \mathsf{rec}(\overline{z}, \overline{v}).P \end{array}$$

where i is an index variable, j an integer, $\overline{u}, \overline{v}$ are patterns, and $\overline{n}$ (resp. $\overline{x}$) is a name (resp. variable) pattern. For sake of clarity, we simply write $!^{\mathsf{i}_1}$ instead of $!^{\mathsf{i}_1 \geq 1}$, and we omit "$\mathsf{else}\ Q$" when $Q = 0$. We also use $\mathsf{let}\ x = u\ \mathsf{in}\ P$ as syntactic sugar for $P\{x \mapsto u\}$, *i.e.,* P in which occurrence of x has been replaced by u.

Example 3. The process $!\mathsf{new}\ n_1.!\mathsf{new}\ n_2.\mathsf{out}(c, \langle n_1, n_2 \rangle)$ in applied pi calculus becomes $!^{\mathsf{i}_1}\ \mathsf{new}\ n_1[\mathsf{i}_1].!^{\mathsf{i}_2}\ \mathsf{new}\ n_2[\mathsf{i}_1, \mathsf{i}_2].\mathsf{out}(c[], \langle n_1[\mathsf{i}_1], n_2[\mathsf{i}_1, \mathsf{i}_2] \rangle)$.

The applied pi calculus is very convenient to model memoryless protocols. However, in this calculus, it is very difficult to express protocols that rely on logs. To address this issue, we add the construction $\mathsf{rec}(z, v)$ to represent the record of the term v in the log through the variable z. The record message construct $\mathsf{rec}(z, v).P$ introduces the possibility to log special entries. Intuitively, this construct will be used to allow a participant to record some information which he may later use to perform some tests.

As usual, names and variables have scopes that are delimited by restrictions, inputs, and rec contructs. We respectively write $\overline{fv}(P), \overline{bv}(P), \overline{fn}(P)$ and $\overline{bn}(P)$ for the sets of *free variables pattern*, *bound variables pattern*, *free names pattern*, and *bound names pattern* of a process P. We assume that processes are *name pattern and variable pattern distinct*, *i.e.,*

- $\overline{bn}(P) \cap \overline{fn}(P) = \overline{bv}(P) \cap \overline{fv}(P) = \emptyset$, and
- any name pattern and variable pattern is at most bound once.

Moreover, a variable that is bound by a rec construct can only occur once in the process, and each bang operator is annotated with a distinct index variable.

Example 4. The process $P_{\mathsf{CA}} := \ !^{\mathsf{i}_1 \geq 1}\ \mathsf{in}(c, x[\mathsf{i}_1]).\ \mathsf{rec}(z_{\mathsf{log}}[\mathsf{i}_1], x[\mathsf{i}_1])$ models an agent, *e.g.,* a certificate authority, who logs all the messages that he receives on channel c. We have $\overline{fn}(P_{\mathsf{CA}}) = \{c\}$, $\overline{bv}(P_{\mathsf{CA}}) = \{x[\mathsf{i}_1], z_{\mathsf{log}}[\mathsf{i}_1]\}$, and $\overline{fv}(P_{\mathsf{CA}}) = \emptyset$.

2.3 Semantics

The semantics is given by a relation defined over *configurations.*

Definition 2. *A* configuration *is a tuple* $(\mathcal{E}; \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log})$ *where:*

- $\mathcal{E}$ *is a set of names;*
- $\mathcal{P}$ *is a multiset of processes such that* $\overline{fv}(P) = \emptyset$ *for any* $P \in \mathcal{P}$*; and*
- $\Phi_{\mathcal{A}}$ *and* Φ_{log} *are sequences of the form* $[w_1 \rhd u_1, \ldots, w_n \rhd u_n]$ *where* $u_1, \ldots, u_n$ *are ground terms and* $w_1, \ldots, w_n$ *are variables.*

The set $\mathcal{E}$ represents the names that are unknown by the attacker; $\Phi_{\mathcal{A}}$ represents the messages that have been sent on some public channels, and that are known by the attacker; whereas Φ_{log} represents the messages that have been stored (*e.g.,* in a log file) by some participants during the execution of the protocol. Such a sequence Φ can be seen as a substitution and we denote $\mathrm{dom}(\Phi)$ its domain. Note that the two sequences of messages may have some messages in common. To model a message that is stored in the log and given to the attacker, we have to use both the rec and the out constructs in the process. Configurations are denoted A, B, *etc*, and we write $fn(A)$ (resp. $bn(A)$) the set of free (resp. bound) names of a configuration A. Given a process P, sometimes we simply write P instead of $(\emptyset; \{P\}; \emptyset; \emptyset)$.

We now define the relation $\xrightarrow{\ell}$ between configurations where ℓ is either an input, an output or a silent action (see below). Note that the sent messages are exclusively stored in the frame $\Phi_{\mathcal{A}}$ and not in the labels (the outputs are made by "reference"), whereas the messages stored in the logs (through the rec construct) are stored in Φ_{log} and are not visible by the attacker (silent action).

Then
$$(\mathcal{E}; \{\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\tau} (\mathcal{E}; \{P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \qquad \text{if } u =_{\mathsf{E}} v$$
Else
$$(\mathcal{E}; \{\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\tau} (\mathcal{E}; \{Q\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \qquad \text{if } u \neq_{\mathsf{E}} v$$
Comm
$$(\mathcal{E}; \{\mathsf{in}(u,x).P; \mathsf{out}(v,u').Q\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\tau} (\mathcal{E}; \{P\{^{u'}/_{x}\}; Q\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \qquad \text{if } u =_{\mathsf{E}} v$$
Out
$$(\mathcal{E}; \{\mathsf{out}(u,v).P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\nu w.\mathsf{out}(M,w)} (\mathcal{E}; \{P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}} \uplus [w \rhd v]; \Phi_{log})$$
$$\text{if } M\Phi_{\mathcal{A}} =_{\mathsf{E}} u,\ \mathit{fn}(M) \cap \mathcal{E} = \emptyset,\ \mathit{fv}(M) \subseteq \mathrm{dom}(\Phi_{\mathcal{A}}) \text{ and } w \text{ is a fresh variable}$$
Input
$$(\mathcal{E}; \{\mathsf{in}(u,x).P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\mathsf{in}(M,N)} (\mathcal{E}; \{P\{^{v}/_{x}\}\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log})$$
$$\text{if } M\Phi_{\mathcal{A}} =_{\mathsf{E}} u,\ \mathit{fn}(M) \cap \mathcal{E} = \emptyset,\ \mathit{fv}(M) \subseteq \mathrm{dom}(\Phi_{\mathcal{A}})$$
$$\text{if } N\Phi_{\mathcal{A}} =_{\mathsf{E}} v,\ \mathit{fn}(N) \cap \mathcal{E} = \emptyset,\ \mathit{fv}(N) \subseteq \mathrm{dom}(\Phi_{\mathcal{A}})$$
Record
$$(\mathcal{E}; \{\mathsf{rec}(z,u).P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\tau} (\mathcal{E}; \{P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log} \uplus [z \rhd u])$$
Repl $$(\mathcal{E}; \{!^{\mathsf{i} \geq j}\, P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\tau} (\mathcal{E}; \{!^{\mathsf{i} \geq j+1}\, P; P\{^{j}/_{\mathsf{i}}\}\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log})$$
New $$(\mathcal{E}; \{\mathsf{new}\ n[i_1,...,i_k].P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\tau} (\mathcal{E} \cup \{n[i_1,...,i_k]\}; \{P\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log})$$
Par $$(\mathcal{E}; \{P \mid Q\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log}) \xrightarrow{\tau} (\mathcal{E}; \{P; Q\} \uplus \mathcal{P}; \Phi_{\mathcal{A}}; \Phi_{log})$$

Then, the relation $\xrightarrow{\ell_1 \dots \ell_n}$ between configurations is defined in the usual way. Given a sequence tr of observable actions tr, we write $A \overset{\mathsf{tr}}{\Rightarrow} B$ when there exists a sequence $\ell_1, \dots, \ell_n$ such that $A \xrightarrow{\ell_1 \dots \ell_n} B$ and tr is obtained from $\ell_1 \dots \ell_n$ by erasing all occurrences of τ.

2.4 Tests performed by users

As previously mentioned, we consider that participants may store some messages in their own private log during the execution of the protocol, and can then verify some properties on their log. For sake of simplicity, we model one global log using Φ_{log}, and we assume that participants only access to the part of the log that is public, and to the values stored in their private log.

We consider formulas built upon elementary formulas, mainly equations and disequations between terms, of the form $M =^? N$ and $M \neq^? N$, and that use classical connectives (*e.g.,* $\wedge$, $\vee$, $\Rightarrow$, $\exists$, ...). In particular, we consider that indices can be existentially and universally quantified over $\mathbb{N}$. Given a log Φ_{log}, we write:

- $\Phi_{log} \models M =^? N$ when $\mathit{fv}(M,N) \subseteq \mathrm{dom}(\Phi_{log})$ and $M\Phi_{log} =_{\mathsf{E}} N\Phi_{log}$.
- $\Phi_{log} \models M \neq^? N$ when $\mathit{fv}(M,N) \subseteq \mathrm{dom}(\Phi_{log})$ and $M\Phi_{log} \neq_{\mathsf{E}} N\Phi_{log}$.

We consider that this grammar of logical formulas on the log Φ_{log} is powerful enough to express most of the properties that could be desired to be verified by participants of a protocol. One might still want to extend this grammar given the needs of specific protocols, but so far, we do not have examples of protocols that could not be expressed in our model.

Example 5. Consider $\phi = \forall i \in \mathbb{N}.\forall j \in \mathbb{N}.\big(z_{start}[i,j] =^? \mathsf{true} \Rightarrow z_{end}[i,j] \neq^? \mathsf{error}\big)$. This formula expresses that for every pair of integers (i,j) that corresponds to a session that has been launched, *i.e.,* $z_{start}[i,j]$ is in the domain of the log and is equal to the constant true, then the session has ended without any error, *i.e.,* the variable $z_{end}[i,j]$ is in the domain too and is different from the constant error.

Given a configuration A and a formula ϕ, we define the set of traces of A that satisfy ϕ as follows:

$$\mathsf{trace}(A,\phi) = \{(\mathsf{tr}, \mathsf{new}\,\mathcal{E}.\Phi_{\mathcal{A}}) \mid A \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{E};\mathcal{P};\Phi_{\mathcal{A}};\Phi_{log}) \text{ and } \Phi_{log} \models \phi\}$$

This is in line with the definition of trace proposed in *e.g.,* [7] when $\phi = \mathsf{true}$.

3 Certificate transparency protocol

To ensure the authenticity of the public keys used in cryptographic protocols, the X.509 public key infrastructure (X.509-PKI) introduced the notion of certificate authorities. A public key is considered as authentic if a certificate authority is able to provide a valid certificate for such public key. However, recent attacks [15,16] showed the weakness of this public key infrastructure.

Indeed, by blindly trusting certificate authorities, one allows a malicious certificate authority to provide fake certificates to any client.

The Certificate Transparency (CT) protocol, proposed by Laurie, Kasper and Langley [12], aims to remove the requirement to trust the certificate authorities by making certificate management transparent. The main idea behind the protocol is not to prevent a certificate authority to misbehave but to be able to detect when a certificate authority did misbehave. In this section, we first describe the protocol and the tests that are supposed to be satisfied by the logs in order to ensure the security properties that the protocol is supposed to achieve.

3.1 Description of the protocol

The protocol relies on public append-only logs in which certificate authorities are compelled to write information that will allow a user or a monitor to verify that they behave properly. In particular, certificate authorities will be required to provide *proofs* to anyone who desires to verify the content of the log. For this extent, the CT protocol relies on Merkle trees [13] as structure for providing proofs. Intuitively, a Merkle tree is a binary tree whose nodes are labeled by the hash of the labels of his children, and where the leaves are labeled with the data of the logs, *i.e.,* certificates in this case. Merkle trees enable one to efficiently prove presence of data in the log, and to prove that the log is maintained append only. These proofs can also be done with a simpler data structure, namely, hash chains, although the proofs are less efficient in that case. Since Merkle trees and hash chains are equivalent from the point of view of the proofs (differing only in terms of efficiency), we choose, for sake of simplicity, to model *hash chains*.

Example 6. Consider certificates c_1, c_2, and c_3. A log file composed of c_1, then c_2, followed by c_3 will be accompanied with the hash chain $\mathsf{h}(c_3, \mathsf{h}(c_2, \mathsf{h}(c_1, \bot)))$.

Typically, this hash whose purpose is to represent the current state of a log will be displayed and given to any participant accessing the log. This will allow him to verify that a certificate is indeed in a log (called *proof of presence*), or that the current log is an extension of a previous one (called *proof of extension*).

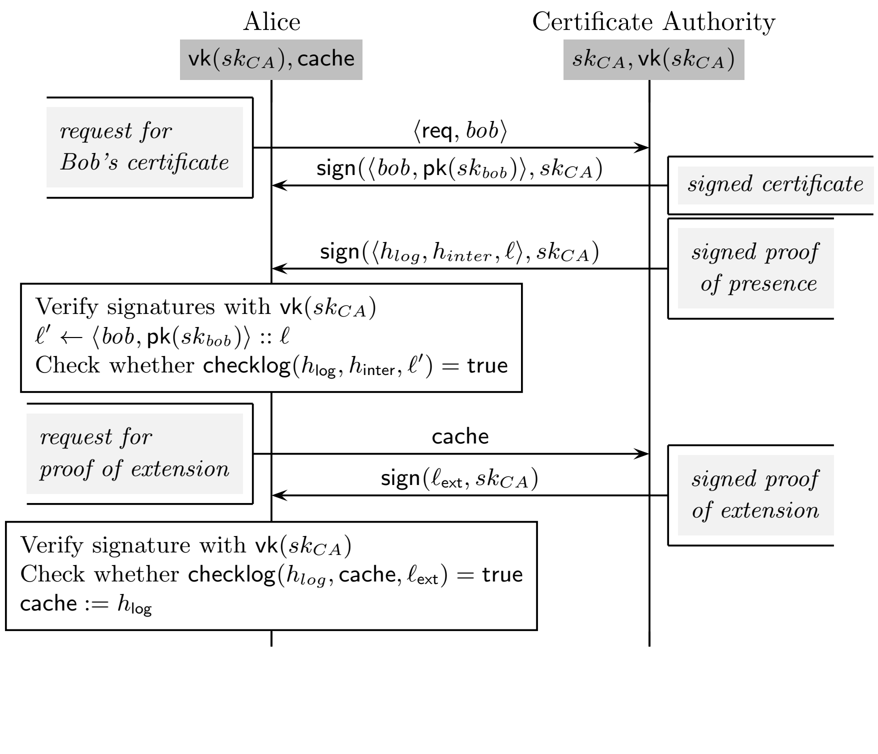
Proof of presence and proof of extension. The proof of presence of a certificate c inside a hash chain h_{log} can be done by giving the hash h_{inter} representing the state of the log before the certificate c was added together with the list $[c_1, \ldots, c_n]$ of certificates corresponding to those that have been added in the log after the certificate c. With these elements, it is now easy to reconstruct h_{log} from h_{inter}. This ensures the presence of c (and also of each c_i) in the hash h_{log}. More formally, we consider a function symbol $\mathsf{checklog}$ that satisfies the following equations:

$$\mathsf{checklog}(z, z, \bot) = \mathsf{true} \quad \mathsf{checklog}(z, z', x{::}y) = \mathsf{checklog}(z, \mathsf{h}(x, z'), y)$$

Intuitively, $\mathsf{checklog}(h_{\mathsf{log}}, h_{\mathsf{inter}}, [c, c_1, \ldots, c_n])$ is true when h_{log} can be derived from h_{inter} by adding c, c_1, $\ldots$, c_n in this order.

In this setting, a proof of extension is similar to a proof of presence. Indeed, proving that h_{log} is an extension of h_{inter} can be done by giving the certificates that allow one to reconstruct h_{log} from h_{inter}.

The request protocol described below is used when Alice wants to obtain (through a certificate authority) the public key of Bob.



Alice
$\mathsf{vk}(sk_{CA}), \mathsf{cache}$
Certificate Authority
$sk_{CA}, \mathsf{vk}(sk_{CA})$
request for Bob's certificate
$\langle \mathsf{req}, bob \rangle$
$\mathsf{sign}(\langle bob, \mathsf{pk}(sk_{bob}) \rangle, sk_{CA})$
signed certificate
$\mathsf{sign}(\langle h_{log}, h_{inter}, \ell \rangle, sk_{CA})$
signed proof of presence
Verify signatures with $\mathsf{vk}(sk_{CA})$
$\ell' \leftarrow \langle bob, \mathsf{pk}(sk_{bob}) \rangle :: \ell$
Check whether $\mathsf{checklog}(h_{\mathsf{log}}, h_{\mathsf{inter}}, \ell') = \mathsf{true}$
request for proof of extension
cache
$\mathsf{sign}(\ell_{\mathsf{ext}}, sk_{CA})$
signed proof of extension
Verify signature with $\mathsf{vk}(sk_{CA})$
Check whether $\mathsf{checklog}(h_{log}, \mathsf{cache}, \ell_{\mathsf{ext}}) = \mathsf{true}$
$\mathsf{cache} := h_{\mathsf{log}}$

We assume that Alice has a *cell* denoted cache that allows her to store some information. In particular, this cell will be used to store the previous hash value she received from the certificate authority. Note that even though cells are not part of our calculus, it is possible to model them using private channels. However, for sake of clarity, we will keep the notation of cells.

Alice first sends a request to the certificate authority to ask for the public key of Bob. The certificate authority answers with the certificate of Bob, composed in fact with Bob's name and public key, signed with his own private signing key, *i.e.,* $\mathsf{sign}(\langle bob, \mathsf{pk}(sk_{bob})\rangle, sk_{CA})$. Moreover, the certificate authority sends several elements, signed with its private key, that will allow the participant to be sure that the certificate for Bob's public key is indeed in the hash chain h_{log} (proof of presence). After performing the checks, Alice will ask the certificate authority the elements to be sure that the current hash chain, *i.e.,* h_{log}, is indeed an extension of the previous hash that Alice has stored in cache (proof of extension). If so, she stores the updated value of the hash chain in her cell cache.

The registering protocol is used when Bob wants to register his public key to the certificate authority. Actually, this protocol is very similar to the previous protocol between Alice and the certificate authority. Bob will send first a request to register his public key by sending the message $\langle \mathsf{reg}, bob, \mathsf{pk}(sk_{bob})\rangle$, and then he will perform the tests as in the request protocol (messages 3, 4, and 5).

The verification process. The purpose of the public logs is to allow anyone to check the log and so to detect any possible misbehaviour of the certificate authority. In addition to the checks done by the users when requesting and registering a public key, some further checks have to be performed.

Check 1. To prevent a certificate authority of binding false public keys to a participant, each participant has to ensure that all the keys associated to him in the log are indeed public keys for which he asked a registration.

Check 2. It is important for the security of the protocol to consider that the users can obtain the current hash value of the log from a different source than the certificate authority itself. Otherwise, the authority could provide different values of the log to different users without them being able to detect anything. The goal here is to ensure that all users have the same (up to some extension) hash value of the public log. Google is exploring the possibility of implementing a gossip protocol to allow users to directly share information. One could also imagine the existence of servers mirroring the public logs as alternative sources for the hash value. In this paper, we abstract ourselves from the implementation and we simply propose an abstract test to ensure that users have the same hash value for the public log (up to some extension).

3.2 The protocol in our calculus

The modelling of the protocol follows the informal description given in Section 3.1. We only present some elements in this section. Note that, in our model, the certificate authority is not assumed to be trustworthy, and thus we do not really need to model it. We simply have to model the fact that the authority has to write information on a public log. Thus, the process modelling the protocol CT is made of three components:

$$P_{\mathsf{CT}} \;=\, !^{\mathsf{r}}\, \mathsf{in}(c,x).\mathsf{rec}(zlog[r],x) \;\mid\; !^{\mathsf{i}}\, (!^{\mathsf{j}}\, \mathsf{new}\, sk_{agent}[\mathsf{i},\mathsf{j}].P_{\mathsf{Reg}}[\mathsf{i},\mathsf{j}] \;\mid\; !^{\mathsf{i}'}\, !^{\mathsf{k}}\, P_{\mathsf{Req}}[\mathsf{i},\mathsf{i}',\mathsf{k}])$$

The process $\mathsf{in}(c,x).\mathsf{rec}(zlog[r],x)$ corresponds to the authority writing on a public log represented by the variables $zlog[r]$. The process $P_{\mathsf{Reg}}[\mathsf{i},\mathsf{j}]$, partially described below, models a session during which the agent $agent[\mathsf{i}]$ registers a new public key $\mathsf{pk}(sk_{agent}[\mathsf{i},\mathsf{j}])$. Note that a given agent $agent[\mathsf{i}]$ may register several public keys. A variable pattern, denoted $zlog^{reg}_{agent}$, is used to store the hash chains of the public log that are sent by the certificate authority.

$$P_{\mathsf{Reg}}[\mathsf{i},\mathsf{j}] = \begin{cases} \mathsf{out}(c,\langle \mathsf{reg}, agent[i], \mathsf{pk}(sk_{agent}[\mathsf{i},\mathsf{j}])\rangle). & \textit{(* register and log} \\ \mathsf{rec}(pk_{agent}[\mathsf{i},\mathsf{j}], \mathsf{pk}(sk_{agent}[\mathsf{i},\mathsf{j}])). & \textit{his own public key *)} \\ \ldots & \\ \mathsf{rec}(zlog^{reg}_{agent}[\mathsf{i},\mathsf{j}], h_{log}). & \end{cases}$$

The process $P_{\mathsf{Req}}[\mathsf{i},\mathsf{i}',\mathsf{k}]$ models a session during which the agent $agent[\mathsf{i}]$ requests the public key of another agent $agent[\mathsf{i}']$. Such a request may happen several times. This is modeled using the parameter k. A variable pattern, denoted $zlog^{req}_{agent}$, is used to store the hash chains of the public log that is sent by the certificate authority. The last line is used to model that a fresh secret is sent on the public channel c using the Bob's public key (the one obtained through the certificate authority).

$$P_{\mathsf{Req}}[\mathsf{i},\mathsf{i}',\mathsf{k}] = \begin{cases} \mathsf{out}(c,\langle \mathsf{req}, agent[\mathsf{i}']\rangle). & \textit{(* request } agent[\mathsf{i}']\textit{'s certificate *)} \\ \ldots & \\ \mathsf{rec}(zlog^{req}_{agent}[\mathsf{i},\mathsf{i}',\mathsf{k}], h_{log}). & \\ \mathsf{new}\, s[\mathsf{i},\mathsf{i}',\mathsf{k}].\, \mathsf{out}(c, \mathsf{aenc}(s[\mathsf{i},\mathsf{i}',\mathsf{k}], pk_b)) & \end{cases}$$

We now detail the tests performed by users. First, each participant $agent[\mathsf{i}]$ has to check that any public keys bound to his name are indeed public keys for which he asked a registration. Intuitively, $\phi_{keys}(\mathsf{i})$ holds if each key pertaining to $agent[\mathsf{i}]$ in the CT log is indeed one of $agent[\mathsf{i}]$'s keys.

$$\phi_{keys}(\mathsf{i}) = \forall \mathsf{r} \in \mathbb{N}. \begin{cases} \mathsf{proj}_1(zlog[\mathsf{r}]) =^? agent[\mathsf{i}] \\ \qquad\qquad \Rightarrow (\exists \mathsf{j} \in \mathbb{N}.\, \mathsf{proj}_2(zlog[\mathsf{r}]) =^? \mathsf{pk}(sk_{agent}[\mathsf{i},\mathsf{j}])) \end{cases}$$

Next, participants must perform a test to ensure that they are tracking the same version of the public log as each other. We encode this test by specifying that the agent checks that each hash value it received during the registration and request protocols is indeed the value of some edition of the log (up to some extension). Thus, we define ϕ_{track} as follows:

$$\begin{aligned} \phi_{track}(\mathsf{i}) = {} & \forall \mathsf{j} \in \mathbb{N}.\, \exists \mathsf{r} \in \mathbb{N}. \big(zlog^{reg}_{agent}[\mathsf{i},\mathsf{j}] =^? \mathsf{h}(zlog[\mathsf{r}], \mathsf{h}(\ldots, \mathsf{h}(zlog[1], \bot))) \\ & \wedge \forall \mathsf{i}', \mathsf{k} \in \mathbb{N}.\, \exists \mathsf{r} \in \mathbb{N}. \big(zlog^{req}_{agent}[\mathsf{i},\mathsf{i}',\mathsf{k}] =^? \mathsf{h}(zlog[\mathsf{r}], \mathsf{h}(\ldots, \mathsf{h}(zlog[1], \bot))) \end{aligned}$$

For example, if both formulas $\phi_{track}(1)$ and $\phi_{track}(2)$ are true, then it means that the agents $agent[1]$ and $agent[2]$ share the same hash values for the public log (up to some extension).

3.3 Security properties: secrecy

The CT protocol was developed to improve the management of public keys. The main security property that this protocol is supposed to ensure is the secrecy of any message that a user could encrypt with the requested public key. This can be expressed as usual as the non-deducibility of the given term for any scenario. However, the usual secrecy property that one can find in the literature is in fact not satisfied by the protocol.

Consider for example that Alice wants to talk to Bob. When Alice asks the certificate for Bob's public key, the attacker can always adds a fake public key in the log associated to Bob and sends this public key to Alice. From the point of view of Alice, the protocol will succeed flawlessly. Hence, Alice will encrypt her secret message with the fake public key created by the certificate authority. The secrecy of the message will be broken. As previously mentioned, the CT protocol does not prevent such man-in-the-middle attack but it will leave some traces that this attack occurred. Thus, we have to adapt the definition of secrecy to include additional tests that the participants will apply afterwards.

Definition 3 (ϕ-testable secrecy). *The CT protocol satisfies ϕ-testable secrecy of $\overline{s}$ if for all $\mathsf{i}_0, \mathsf{i}'_0, \mathsf{j}_0 \in \mathbb{N}$, for all $(\mathsf{tr}, \mathsf{new}\,\mathcal{E}.\Phi) \in \mathsf{trace}(P_{\mathsf{CT}}, \phi(\mathsf{i}_0, \mathsf{i}'_0, \mathsf{j}_0))$, there is no M such that $fn(M) \cap \mathcal{E} = \emptyset$, $fv(M) \subseteq \mathrm{dom}(\Phi)$ and $M\Phi =_{\mathsf{E}} s[\mathsf{i}_0, \mathsf{j}_0, \mathsf{i}'_0]$.*

Intuitively, a protocol satisfies ϕ-testable secrecy as soon as the traces of the protocol under study that satisfy the formula ϕ do not reveal the secret s. All traces that do not satisfy ϕ are discarded. Note that when considering the formula $\phi_0 = \mathsf{true}$, the ϕ_0-testable secrecy is in fact the usual secrecy property of the literature (no trace is discarded). We have seen that the P_{CT} protocol does not satisfy the secrecy property in this case. We may consider a formula ϕ_1 which represents the fact that everyone apply their own tests:

$$\phi_1(\_, \_, \_) = \forall \mathsf{i} \in \mathbb{N}. \big(\phi_{keys}(\mathsf{i}) \wedge \phi_{track}(\mathsf{i})\big)$$

Note that the trace corresponding to the previous attack does not satisfy this test since Bob will see that the certificate authority added a fake certificate to his name. Thus, this trace is discarded when checking the ϕ_1-testable secrecy on the CT protocol. We can actually state a more general security property that requires only some checks by the participants who exchanged the secret. This is the purpose of the following formula ϕ_2:

$$\phi_2(\mathsf{i}_0, \mathsf{i}'_0, \_) = \phi_{track}(\mathsf{i}_0) \wedge \phi_{track}(\mathsf{i}'_0) \wedge \phi_{keys}(\mathsf{i}'_0).$$

$\phi_{track}(\mathsf{i}_0) \wedge \phi_{track}(\mathsf{i}'_0)$ models the fact that the participants $agent[\mathsf{i}_0]$ and $agent[\mathsf{i}'_0]$ have to be synchronised, and $\phi_{keys}(\mathsf{i}'_0)$ models the fact that $agent[\mathsf{i}'_0]$ has to check that only his keys are bound to his name in the public log. Thus the ϕ_2-testable secrecy ensures the secrecy of the secret sent by $agent[\mathsf{i}_0]$ to $agent[\mathsf{i}'_0]$ if the two participants have done their tests. It also means that even if others participants did not check their tests or even if the certificate authority misbehaved towards other participants, the secret between $agent[\mathsf{i}_0]$ and $agent[\mathsf{i}'_0]$ is still secure.

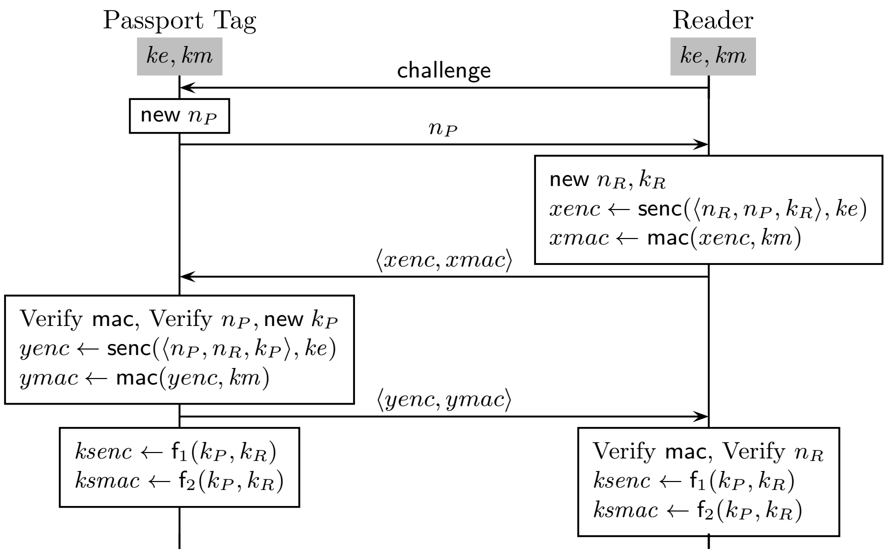
These examples show the usefulness of different kinds of test. It still remains to be formally demonstrated that the properties hold when the tests are true, but that requires some tool support.

4 The e-passport application

To illustrate the usefulness of our approach on the analysis of privacy-type properties, we consider the e-passport application, and we review the existing linkability attack that exists on the French version of the BAC protocol [3]. In particular, we will explain how the holder of such an e-passport who has some concerns with his privacy, can at least detect if he has been attacked.

4.1 Protocol description

To describe and model this protocol, we consider the function symbols senc, sdec, $\langle\ \ \rangle$, proj_1, proj_2, mac, f_1, f_2, and the equations $\mathsf{sdec}(\mathsf{senc}(x, y), y) = x$, $\mathsf{proj}_1(\langle x_1, x_2\rangle) = x_1$, and $\mathsf{proj}_2(\langle x_1, x_2\rangle) = x_2$ to take into account the algebraic properties of these operators.


Passport Tag
Reader
ke, km
ke, km
challenge
new n_P
n_P
new n_R, k_R
$xenc \leftarrow \mathsf{senc}(\langle n_R, n_P, k_R\rangle, ke)$
$xmac \leftarrow \mathsf{mac}(xenc, km)$
$\langle xenc, xmac\rangle$
Verify mac, Verify n_P, new k_P
$yenc \leftarrow \mathsf{senc}(\langle n_P, n_R, k_P\rangle, ke)$
$ymac \leftarrow \mathsf{mac}(yenc, km)$
$\langle yenc, ymac\rangle$
$ksenc \leftarrow \mathsf{f}_1(k_P, k_R)$
$ksmac \leftarrow \mathsf{f}_2(k_P, k_R)$
Verify mac, Verify n_R
$ksenc \leftarrow \mathsf{f}_1(k_P, k_R)$
$ksmac \leftarrow \mathsf{f}_2(k_P, k_R)$

The reader first asks for a challenge, and the passport answers to this request by sending a fresh nonce n_P. The reader will then encrypt n_P together with a fresh nonce of his own (n_R) using the shared key ke, and mac everything with the key km. The keys ke and km are derived from some information printed on the passport, which has, in theory, been scanned before the wireless communication begins. At this point, the passport checks the mac, and then decrypts the ciphertext to verify whether his own nonce n_P is indeed inside. The keys $ksenc$ and $ksmac$ are then derived from the keys k_R and k_P that have been exchanged.

So far, this description holds for the English and the French version of this protocol. However, the ICAO does not specify what kind of error messages the passport should return when an operation fails. Actually, a French passport responds to an incorrect MAC with the error code 6300, which means no information given. If the MAC is correct, and the passport went on to find that the nonce did not match then it responds with an error code 6A80, meaning incorrect parameters. This actually leads to a real life attack on anyone carrying a French e-passport [3].

Description of the attack. After listening to one session of the BAC protocol between the targeted passport and an honest reader, the attacker in presence of an unidentified passport, will be able to replay the message:

$$\langle xenc, \mathsf{mac}(xenc, km)\rangle \text{ where } xenc = \mathsf{senc}(\langle n_R, n_P, k_R\rangle, ke).$$

The unidentified passport will answer with an error message. This message will be either the error code 6300 or the error code 6A80. This code allows the attacker to know whether the passport in presence is the targeted passport.

4.2 Security properties: unlinkability

The purpose of the BAC protocol is to establish a fresh session that will be used to protect the personnal data before sending those information to the reader. A typical scenario can be modeled using the process:

$$P_{\mathsf{BAC}} = !^{\mathsf{i}}\ \mathsf{new}\ ke[\mathsf{i}].\mathsf{new}\ km[\mathsf{i}].!^{\mathsf{j}}\ (P_{\mathsf{Pass}}[\mathsf{i},\mathsf{j}] \mid P_{\mathsf{Reader}}[\mathsf{i},\mathsf{j}])$$

The two replications model several passports lauching possibly several instances of the protocol. Moreover, the process $P_{\mathsf{Reader}}[\mathsf{i},\mathsf{j}]$ is used to model the operations done by readers already sharing the private keys $ke[\mathsf{i}], km[\mathsf{i}]$ of the passport (*e.g.* through optical reading).

To formalize unlinkability, we annotate processes as it was done in [3]. In particular, we label the actions of our processes to know *e.g.,* whether two actions have been performed by the same passport, or in the same session. In our framework, this can be done by annotating each visible action with a distinct label parametrized with the index variables occurring in the replication above this action.

Example 7. For a process $!^{\mathsf{i}}\ \mathsf{out}(c, ok).!^{\mathsf{j}}\ \mathsf{in}(c, x[\mathsf{i},\mathsf{j}])$, the augmented process is $!^{\mathsf{i}}\ \mathsf{out}^{\ell[\mathsf{i}]}(c, u).!^{\mathsf{j}}\ \mathsf{in}^{\ell'[\mathsf{i},\mathsf{j}]}(c, x[i,j])$ where ℓ and ℓ' are two different labels, *i.e.,* constants, parametrized with i and i,j respectively.

The semantics of our processes would also display these annotations. Now, given such a sequence of annotated actions we can associate a sequence of actions and a sequence of annotations. Given an annotated trace tr, we denote by $\mathsf{act}(\mathsf{tr})$ the sequence of actions, and by $\mathsf{ann}(\mathsf{tr})$ the sequence of annotations. We will denote $\tilde{P}$ the augmented process obtained from P.

Relying on these annotations, we can now define the notion of ϕ-testable unlinkability.

Definition 4 (ϕ-testable unlinkability). *We say that P_{BAC} satisfies ϕ-testable unlinkability if for all $\mathsf{i}_0, \mathsf{j}_0, \mathsf{j}'_0 \in \mathbb{N}$, for all $(\mathsf{tr}, \varphi) \in \mathsf{trace}(\tilde{P}_{BAC}, \phi(\mathsf{i}_0, \mathsf{j}_0, \mathsf{j}'_0))$ with $\mathsf{ann}(\mathsf{tr}) = \ell_1[i_1, j_1]. \ldots .\ell_n[i_n, j_n]$, for all $k, r \in \{1, \ldots, n\}$, if $i_k = i_r = \mathsf{i}_0$, $\mathsf{j}_0 = j_k$, $\mathsf{j}'_0 = j_r$ and $j_k \neq j_r$ then there exists $(\mathsf{tr}', \varphi') \in \mathsf{trace}(\tilde{P}_{BAC}, \phi(\mathsf{i}_0, \mathsf{j}_0, \mathsf{j}'_0))$ such that $\mathsf{ann}(\mathsf{tr}') = \ell'_1[i'_1, j'_1]. \ldots .\ell'_n[i'_n, j'_n]$ with $\mathsf{act}(\mathsf{tr}) = \mathsf{act}(\mathsf{tr}')$, $i'_k \neq i'_r$, and $\varphi \sim \varphi'$.*

Assuming $\phi = \mathsf{true}$, this definition is in line with the original definition proposed in [3], and not satisfied by the French version of the passport. Intuitively, the previous definition indicates that for any trace of the protocol where passports can execute several sessions, if two actions are executed by the same passport during two different sessions then there exists an other equivalent trace where these two actions are executed by two different passports.

The main advantage of this definition is that it also allows us to state a weaker form of unlinkability. We can express the unlinkability of a set of French e-passports that do not emit the error Error6A80. For this, assuming the process modelling the role of the passport has been modified to store in its log when it starts and ends a session, and also the error code that it has emitted so far. In other words, we assume here that each owner of a passport has a private log on which he can write. One can assume for example the presence of a small device that listens messages sent by the passport and writes messages on logs.

We can thus consider the following test:

$$\phi_1(\_,\_,\_) = \forall \mathsf{i} \in \mathbb{N}.\forall \mathsf{j} \in \mathbb{N}.(z_{\mathsf{start}}[\mathsf{i},\mathsf{j}] =^? \mathsf{true} \Rightarrow z_{\mathsf{end}}[\mathsf{i},\mathsf{j}] \neq^? \mathsf{Error6A80}).$$

However, we may also be interested to state an unlinkability property w.r.t. to a given passport i_0 without saying anything on the other ones. This can be achieved by considering the following test:

$$\phi_2(\mathsf{i}_0,\_,\_) = \forall \mathsf{j} \in \mathbb{N}.(z_{\mathsf{start}}[\mathsf{i}_0,\mathsf{j}] =^? \mathsf{true} \Rightarrow z_{\mathsf{end}}[\mathsf{i}_0,\mathsf{j}] \neq^? \mathsf{Error6A80}).$$

Note that the ϕ_2-testable unlinkability ensures the non-tracability of a passport as long as this particular passport never emitted Error6A80. However, we may want to strengthen even more the security property. Indeed, requiring that a passport never emit Error6A80 during all its sessions is quite strong. Actually, we may say something about the unlinkability of two sessions of the same passport instead of all sessions. To do so, we consider the following test:

$$\phi_3(\mathsf{i}_0,\mathsf{j}_0,\mathsf{j}'_0) = z_{\mathsf{end}}[\mathsf{i}_0,\mathsf{j}_0] \neq^? \mathsf{Error6A80} \wedge z_{\mathsf{end}}[\mathsf{i}_0,\mathsf{j}'_0] \neq^? \mathsf{Error6A80}.$$

The ϕ_3-testable unlinkability ensures that two sessions (here indexed by j_0 and j'_0) of a given passport indexed i_0 cannot be linked if they both did not emit the error Error6A80. Amongst the other security properties, the ϕ_3-testable unlinkability is the strongest property that the French e-passport can satisfy.

5 Conclusion

Many kinds of cloud and online services involve providers which are not trustworthy, but also not fully malicious in the sense that they are willing to offer evidence of their correct behaviour. Such providers could cheat, but doing so would eventually be detected by their customers. We have presented a model and framework to analyse such services, in which users carry out tests on data that have been logged during the execution of the protocol. If these tests succeed, then security properties are assured. For that reason, we call them testable properties.

Currently, no software tool exists that can automatically verify testable properties. Nevertheless, it should be possible to adapt some existing tools to analyse these kind of properties. In particular, among the tools which can check trace equivalence for a bounded number of sessions, the APTE tool [6] can analyse protocols in presence of inequalities. Thus, this framework seems to be particularly well-suited to being generalised to testable properties.

Acknowledgement. The authors acknowledge partial support from the EPSRC projects *Trustworthy Voting Systems* and *Analysing Security and Privacy Properties*, as well as the project JCJC VIP ANR-11-JS02-006.

References

1. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. Theory of Cryptography, 2007.
2. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symp. on Principles of Programming Languages (POPL'01).*
3. M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, 2010.
4. Mayla Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10).* IEEE Computer Society Press, 2010.
5. Sergiu Bursuc, Gurchetan S. Grewal, and Mark Dermot Ryan. Trivitas: Voters directly verifying votes. In *VOTE-ID*, pages 190–207, 2011.
6. V. Cheval. APTE (Algorithm for Proving Trace Equivalence), 2013. `http://projects.lsv.ens-cachan.fr/APTE/`.
7. V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *Proc. 18th ACM Conference on Computer and Communications Security (CCS'11)*, 2011.
8. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. In *Proc. 18th European Symposium on Research in Computer Security (ESORICS'13)*, pages 1–18. Springer, 2013.
9. 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.
10. Gurchetan S. Grewal, Mark Dermot Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. Caveat coercitor: Coercion-evidence in electronic voting. In *IEEE Symposium on Security and Privacy*, pages 367–381, 2013.
11. Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *ESORICS*, pages 389–404, 2010.
12. B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962 (Experimental), 2013.
13. Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378, 1987.
14. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
15. P. Roberts. Phony SSL certificates issued for Google, Yahoo, Skype, others, March 2011. `http://threatpost.com/phony-ssl-certificates-issued-google-yahoo-skype-others-032311`.
16. T. Sterling. Second firm warns of concern after Dutch hack, September 2011. `http://news.yahoo.com/second-firm-warns-concern-dutch-hack-215940770.html`.