Stateful Applied Pi Calculus

Myrto Arapinis, Jia Liu, Eike Ritter, and Mark Ryan

School of Computer Science, University of Birmingham, UK

Abstract. We extend the applied pi calculus with state cells, which are used to reason about protocols that store persistent information. Examples are protocols involving databases or hardware modules with internal state. We distinguish between private state cells, which are not available to the attacker, and public state cells, which arise when a private state cell is compromised by the attacker. For processes involving only private state cells we define observational equivalence and labelled bisimilarity in the same way as in the original applied pi calculus, and show that they coincide. Our result implies Abadi-Fournet's theorem – the coincidence of observational equivalence and labelled bisimilarity – in a revised version of the applied pi calculus. For processes involving public state cells, we can essentially keep the definition of observational equivalence, but need to strengthen the definition of labelled bisimulation in order to show that observational equivalence and labelled bisimilarity coincide in this case as well.

Keywords: applied pi calculus, global state, bisimulation, security protocols.

1 Introduction

Security protocols are small distributed programs that use cryptography in order to achieve a security goal. The complexity that arises from their distributed nature motivates formal analysis in order to prove logical properties of their behaviour; fortunately, they are often small enough to make this kind of analysis feasible. Various logical methods have been used to model security protocols; process calculi have been particularly successful [3, 5, 31]. For example, the TLS protocol used by billions of users every day was analysed using ProVerif [11].

More recently, protocol analysis methods have been applied to stateful protocols – that is, protocols which involve persistent state information that can affect and be changed by protocol runs. Hardware devices that have some internal memory can be described by such protocols. For example, Yubikey is a USB device which generates one-time passwords based on encryptions of a secret ID, a running counter and some random values using a unique AES-128 key contained in the device. The trusted platform module (TPM) is another hardware chip that has a variety of registers which represent its state, and protocols for updating them. Radio-frequency identification (RFID) is a wireless technology for automatic identification and is currently deployed in electronic passports, tags for consumer goods, livestock and pets tracking, etc. An RFID-tag has a small area for storing secrets, which may be modified.

A process calculus can be made to work with such stateful protocols either by extension or by encoding. Extension means adding to the calculus explicit constructs for

M. Abadi and S. Kremer (Eds.): POST 2014, LNCS 8414, pp. 22–41, 2014.

working with the stateful aspects, while encoding means using combinations of the primitives that already exist. Encodings have the advantage that they keep the calculus simple and elegant, but (as argued in [3]) there may not be encodings for all the aspects we want, and in cases that encodings exist they may not be suitable for the analysis of security properties. StatVerif [7] demonstrates this: a natural way of encoding state using restricted channels prevents ProVerif from proving security. ProVerif also provides some built-in features, such as tables and phases, which provide only limited ways for modelling states. In particular, tables are defined as predicates which allow processes to store data by extending a predicate for the data. Hence there is no notion of the "current" state, and values cannot be deleted from tables. Phases are used to model the protocols with several stages. But there can be only finitely many phases, which can only be run in sequence, whereas a state may have infinitely many arbitrary values. Since our starting point is the applied pi calculus [3], we follow the philosophy adopted by its authors, which is to design a calculus that has the right primitives built in.

Our Contributions. We present an extension of the applied pi calculus by adding state cells, which are used to reason about protocols that store persistent information. We distinguish between private state cells, which are not available to the attacker, and public state cells, which arise when a private state cell is compromised by the attacker. In our stateful language, a private state cell is guarded by the scope restriction; its access is limited to some designated processes. When a private state cell gets compromised, the cell becomes public and this scenario is modelled by removing the scope restriction of that cell. We first define observational equivalence and labelled bisimilarity for processes having only private state cells, and we prove that two notions coincide as expected. By encoding the private state cells with restricted channels while keeping observational equivalence, our coincidence result can be seen to imply Abadi-Fournet's theorem [3, Theorem 1], in a revised version of applied pi calculus. As far as we can see, the only available proof for this theorem is [28] which is an unpublished manuscript. Despite having no published proof, this theorem has been widely used in many publications, for example [19, 8, 4, 18, 20].

We also discuss an extension of our language with public state cells. The obvious notion of labelled bisimilarity does not capture observational equivalence on public state cells. Designing a labelled bisimilarity on public state cells turns out to be unexpectedly difficult. Public state cells introduce many special language features which are significantly different from private state cells. Moreover, the addition of public state cells increases the capabilities of the attacker significantly. Hence we strengthen the definition of labelled bisimilarity to show that observational equivalence and labelled bisimulation coincide.

As an illustration, we analyse the OSK protocol [26] for RFID tags. We model its untraceability by private state cells and model its forward privacy by public state cells.

Related Work. StatVerif [7] is an extension of ProVerif process language [13] with private state cells. The main contribution there is to extend the ProVerif compiler to a compiler for StatVerif. The security property of interest there is secrecy which is modelled by reachability on the traces. This paper is a fundamental generalisation of the previous StatVerif work. The focus in this paper is to build a stateful language based

on applied pi calculus, explore its language features and discuss indistinguishability, which is modelled by observational equivalence and analysed by labelled bisimilarity.

There are other languages that have been used to model protocols involving persistent state, but they are lower-level languages that are further away than our process language from the protocol design. Strand spaces have been generalised to work with the global state required by a trusted party charged with enforcing fair exchange [25]. The verifier Tamarin [33] uses multi-set rewriting (in which antecedents of applied rules are withdrawn from the knowledge set in order to represent state changes); it has been used to analyse hardware password tokens [27]. Multi-set rewriting is also used in [30], where state changes are important to represent revocation of cryptographic keys. Horn clauses rather than multiset rewriting are used in [22], in order to represent state changes made to registers of the TPM hardware module.

Reasoning about programming languages involving states has been extensively studied (e.g. [34, 23]). There are very strong interactions between programing language features and state, hence the reasoning principles are very specific to the precise combination of features. In this work we build on the work on reasoning principles for process calculi using bisimulation and show how to extend these principles to handle global state.

Outline. The next section defines syntax and semantics for the stateful applied pi calculus. Section 3 discusses the process equivalences and encoding for private state cells, and derives Abadi-Fournet's theorem. Section 4 extends our stateful language with public state cells. The paper concludes in Section 5.

2 Stateful Applied Pi Calculus

In this section, we extend the applied pi calculus [3] with constructs for states, and define its operational semantics. In fact, we do not directly build the stateful language on top of applied pi calculus, because we want to avoid working with the *structural equivalence* relation. More precisely, reasoning about the equivalent classes induced by structural equivalence turns out to be difficult and normally results in long tedious proofs [21, 18, 29, 17]. Our language inherits constructs for scope restriction, communication and active substitutions from applied pi calculus while having multisets of processes and active substitutions makes it possible to specify an operational semantics which does not involve any structural equivalence.

2.1 Syntax

We assume two disjoint, infinite sets $\mathcal{N}$ and $\mathcal{V}$ of *names* and *variables*, respectively. We rely on a sort system including a universal base sort, a cell sort and a channel sort. The sort system splits $\mathcal{N}$ into channel names $\mathcal{N}_{ch}$, base names $\mathcal{N}_b$ and cell names $\mathcal{N}_s$; similarly, $\mathcal{V}$ is split into channel variables $\mathcal{V}_{ch}$ and base variables $\mathcal{V}_b$. Unless otherwise stated, we use a, b, c as channel names, s, t as cell names, and x, y, z as variables. Meta variables u, v, w are used to range over both names and variables.

A signature Σ consists of a finite set of function symbols, each with an arity. A function symbol with arity 0 is a constant. Function symbols are required to take arguments

and produce results of the base sort only. *Terms*, ranged over by M, N, are built up from variables and names by function application:

$M, N ::=$	terms
a, b, c, k, m, n, s	names
x, y, z	variables
$f(M_1, \ldots, M_\ell)$	function application

We write $var(M)$ and $name(M)$ for the variables and names in M, respectively. Tuples such as $u_1 \cdots u_\ell$ and $M_1 \cdots M_\ell$ will be denoted by $\widetilde{u}$ and $\widetilde{M}$, respectively. Terms are equipped with an equational theory $=_\Sigma$ that is an equivalence relation closed under substitutions of terms for variables, one-to-one renamings and function applications.

The grammar for the *plain process* is given below. The operators for nil process 0, parallel composition $|$, replication !, scope restriction νn, conditional `if - then - else` , input $u(x)$ and output $\overline{u}\langle M\rangle$ are the same as the ones in applied pi calculus [3]. A state cell is a special process of the form $[s \mapsto M]$ where s is the cell name and M is the current value of s. The process `lock` $s.P$ locks the cell s for the subsequent process P. When the cell s is locked, another process that intends to access the cell has to wait until the cell is unlocked by a primitive `unlock` s. The process `read` s `as` $x.P$ reads the value in the cell and stores it in x in P. The process $s := M.P$ assigns the value M to the cell and continues as P.

$P, Q, R ::=$	plain process
0	nil process
$P \mid Q$	parallel composition
$!\,P$	replication
$\nu n.P$	name restriction
`if` $M = N$ `then` P `else` Q	conditional
$u(x).P$	input
$\overline{u}\langle M\rangle.P$	output
$[s \mapsto M]$	cell s, containing term M
$s := M.\, P$	writing a cell
`read` s `as` $x.P$	reading a cell
`lock` $s.P$	locking a cell
`unlock` $s.P$	unlocking a cell

subject to the following requirements:

- x, M, N are not of cell sort; $u \in \mathcal{N}_{ch} \cup \mathcal{V}_{ch}$ and $s \in \mathcal{N}_s$; additionally, M is of base sort in both $[s \mapsto M]$ and $s := M.\, P$;
- for every `lock` $s.\, P$, the part P of the process must not include parallel or replication unless it is after an `unlock` s.
- for a given cell name s, the replication operator ! must not occur between νs and $[s \mapsto M]$.

These side conditions rule out some nonsense processes, such as `lock` $s.\,!P$, `lock` $s.\,(P \mid Q)$, $\nu s.![s \mapsto M]$ and $\nu s.([s \mapsto M] \mid [s \mapsto N])$, while keep some reasonable processes, such as `lock` s.`unlock` $s.\,!P$, `lock` s.`unlock` $s.\,(P \mid Q)$ and $!\nu s.[s \mapsto M]$.

An *extended process*, ranged over by A, B, C, is an expression of the form $\nu\widetilde{n}.\,(\sigma, S, \mathcal{P})$ where

- $\nu\widetilde{n}$ is a set of name restrictions;
- σ is a substitution $\{M_1/x_1, \ldots, M_n/x_n\}$ which replaces variables of base sort with terms of base sort; the domain $dom(\sigma)$ of σ is $\{x_1, \ldots, x_n\}$; the domain $dom(\nu\widetilde{n}.(\sigma, S, \mathcal{P}))$ of the extended process $\nu\widetilde{n}.(\sigma, S, \mathcal{P})$ is also $dom(\sigma)$; we require that $dom(\sigma) \cap fv(M_1, \ldots, M_n, \mathcal{P}, S) = \emptyset$;
- $S = \{s_1 \mapsto M_1, \ldots, s_m \mapsto M_m\}$ is a set of state cells such that $s_1, \ldots, s_m$ are pairwise-distinct cell names and terms $M_1, \ldots, M_m$ are of base sort; we write $dom(S)$ for $\{s_1, \ldots, s_m\}$ and $S(s_i)$ for M_i $(1 \leq i \leq m)$;
- $[s \mapsto M]$ can only occur at most once for a given cell name s, and if a cell name s is not restricted by any νs, a state cell $s \mapsto M$ can only occur in S;
- $\mathcal{P} = \{(P_1, L_1), \ldots, (P_k, L_k)\}$ is a multiset of pairs where P_i is a plain process and L_i is a set of cell names; $L_i \cap L_j = \emptyset$ for any $1 \leq i, j \leq k$ and $i \neq j$; for each $s \in L_i$, the part of the process P_i must not include parallel or replication unless it is after a `unlock` s; we write $locks(\mathcal{P})$ for the set $L_1 \cup \cdots \cup L_k$, namely the locked cells in $\mathcal{P}$.

In an extended process $\nu\widetilde{n}.(\sigma, S, \mathcal{P})$, the substitution σ is similar to the active substitutions in applied pi calculus [3] which denote the static knowledge that the process exposes to the environment. A minor difference with [3] is that substitutions here are only defined on terms of base sort which will be explained later. State cells are mutable and the value of a cell may be changed during the running of processes. If a process P locks a cell s, then this status information will be kept as $(P, \{s\} \cup L)$ in $\mathcal{P}$. At any time, the cell s can be locked at most once in $\mathcal{P}$.

The variable x in "$u(x)$" and "`read` s `as` x" are bound, as well as the name n in νn. This leads to the usual notions of bound and free names and variables. We shall use $fn(A)$ for free names, use $fs(A)$ for free cell names, use $fv(A)$ for free variables, use $bn(A)$ for bound names, and use $bv(A)$ for bound variables of A. Let $fnv(A) = fn(A) \cup fv(A)$ and $bnv(A) = bn(A) \cup bv(A)$. Following the conventions in [32], we shall identify processes which are α-convertible. We write "=" for both syntactical equality and equivalence under α-conversion. Captures of bound names and bound variables are avoided by implicit α-conversion.

An extended process $\nu\widetilde{n}.(\sigma, S, \mathcal{P})$ is called *closed* if each variable is either defined by σ or bound, each cell name s is defined by exactly one "$s \mapsto M$" (either in S or in $\mathcal{P}$), and $locks(\mathcal{P}) \subseteq dom(S)$. We may write $(\sigma, S, \mathcal{P})$ for $\nu\emptyset.(\sigma, S, \mathcal{P})$, and write $\nu\widetilde{n}, \widetilde{m}.(\sigma, S, \mathcal{P})$ for $\nu(\widetilde{n} \cup \widetilde{m}).(\sigma, S, \mathcal{P})$.

When we write $\sigma = \sigma_1 \cup \sigma_2$ for some substitution σ or $S = S_1 \cup S_2$ for some state cells S, we assume that $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$ as well as $dom(S_1) \cap dom(S_2) = \emptyset$. For variables $\widetilde{x}$, we define $\sigma_{\setminus\widetilde{x}}$ to be the substitution $\{\, z\sigma/z \mid z \in dom(\sigma) \text{ and } z \notin \widetilde{x} \,\}$. If $A = \nu\widetilde{n}.(\sigma, S, \mathcal{P})$, we write $A_{\setminus\widetilde{x}}$ for $\nu\widetilde{n}.(\sigma_{\setminus\widetilde{x}}, S, \mathcal{P})$.

An *evaluation context* $\nu\widetilde{n}.(\sigma\text{-}, S\text{-}, \mathcal{P}\text{-})$ is an extended process with holes "-" for substitution, state cells and plain processes. Let $\mathcal{C} = \nu\widetilde{n}.(\sigma\text{-}, S\text{-}, \mathcal{P}\text{-})$ be an evaluation context and $A = \nu\widetilde{m}.(\sigma_a, S_a, \mathcal{P}_a)$ be a closed extended process with $\widetilde{m} \cap (\widetilde{n} \cup fn(\sigma, S, \mathcal{P})) = dom(\sigma) \cap dom(\sigma_a) = dom(S) \cap dom(S_a) = \emptyset$. The result of applying

$$\nu\tilde{n}.\,(\sigma, S, \mathcal{P} \cup \{(!\,P, \emptyset)\}) \xrightarrow{\tau} \nu\tilde{n}.\,(\sigma, S, \mathcal{P} \cup \{(!\,P, \emptyset), (P, \emptyset)\})$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(P \mid Q, \emptyset)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(P, \emptyset), (Q, \emptyset)\})$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(\nu m.P, L)\}) \xrightarrow{\tau} \nu\tilde{n}, m.(\sigma, S, \mathcal{P} \cup \{(P, L)\}) \text{ if } m \notin fn(\tilde{n}, \sigma, S, \mathcal{P}, L)$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{([s \mapsto M], \emptyset)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P}) \text{ if } s \in \tilde{n} \text{ and } s \notin dom(S)$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(a(x).P, L_1)\} \cup \{(\overline{a}\langle M\rangle.Q, L_2)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(P\{M/x\}, L_1), (Q, L_2)\}))$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(\texttt{if } M = N \texttt{ then } P \texttt{ else } Q, L)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(P, L)\}) \text{ if } M =_\Sigma N$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(\texttt{if } M = N \texttt{ then } P \texttt{ else } Q, L)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(Q, L)\}) \text{ if } M \neq_\Sigma N \text{ and } var(M, N) = \emptyset$$

$$\nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(\texttt{read } s \texttt{ as } x.P, L)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(P\{M/x\}, L)\})$$
$$\text{if } s \in \tilde{n} \cup L \text{ and } s \notin locks(\mathcal{P})$$

$$\nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(s := N.P, L)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto N\}, \mathcal{P} \cup \{(P, L)\})$$
$$\text{if } s \in \tilde{n} \cup L \text{ and } s \notin locks(\mathcal{P})$$

$$\nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(\texttt{lock } s.P, L)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(P, L \cup \{s\})\})$$
$$\text{if } s \in \tilde{n} \text{ and } s \notin L \cup locks(\mathcal{P})$$

$$\nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(\texttt{unlock } s.P, L)\}) \xrightarrow{\tau} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(P, L \setminus \{s\})\}) \text{ if } s \in \tilde{n} \cap L$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(a(x).P, L)\}) \xrightarrow{a(M)} \nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(P\{M\sigma/x\}, L)\}) \text{ if } name(a, M) \cap \tilde{n} = \emptyset$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(\overline{a}\langle c\rangle.P, L)\}) \xrightarrow{\overline{a}\langle c\rangle} \nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(P, L)\}) \text{ if } a, c \notin \tilde{n}$$

$$\nu\tilde{n}, c.(\sigma, S, \mathcal{P} \cup \{(\overline{a}\langle c\rangle.P, L)\}) \xrightarrow{\nu c.\overline{a}\langle c\rangle} \nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(P, L)\}) \text{ if } a, c \notin \tilde{n} \text{ and } a \neq c$$

$$\nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(\overline{a}\langle M\rangle.P, L)\}) \xrightarrow{\nu x.\overline{a}\langle x\rangle} \nu\tilde{n}.(\sigma \cup \{M/x\}, S, \mathcal{P} \cup \{(P, L)\})$$
$$\text{if } a \notin \tilde{n} \text{ and } M \text{ is of base sort and } x \text{ is fresh}$$

Fig. 1. Operational Semantics

$\mathcal{C}$ to A is an extended process defined by:

$$\mathcal{C}[A] = \nu\widetilde{n}, \widetilde{m}.(\sigma\sigma_a \cup \sigma_a, S\sigma_a \cup S_a, \mathcal{P}\sigma_a \cup \mathcal{P}_a)$$

An evaluation context $\mathcal{C}$ *closes* A when $\mathcal{C}[A]$ is a closed extended process.

2.2 Operational Semantics

The *transition* relation $A \xrightarrow{\alpha} A'$ is the smallest relation on extended processes defined by the rules in Figure 1. The action α is either an internal action τ, an input $a(x)$, an output of channel name $\overline{a}\langle c\rangle$, an output of bound channel name $\nu c.\overline{a}\langle c\rangle$, or an output of terms of base sort $\nu x.\overline{a}\langle x\rangle$. The transitions for conditional branch, communication, sending and receiving channel names and complex messages are typical and essentially the same as the ones in applied pi calculus. In particular, the output $\nu x.\overline{a}\langle x\rangle$ for term M generates an "alias" x for M which is kept in the substitution part of the extended process. As mentioned before, state cells are used to model the hardware or the database to which the access is usually mutually-exclusive. When a state cell is locked, the other process that intends to access the cell must wait until the cell is released.

3 Private State Cells

3.1 Equivalences for Private State Cells

We first discuss observational equivalence and labelled bisimilarity on the *extended processes with only private state cells*, that is, each cell name s occurring in the processes is within the scope of a restriction νs. We will discuss an extension of the language with public state cells in Section 4.

Observational equivalence [3] has been widely used to model properties of security protocols. It captures the intuition of indistinguishability from the attacker's point of view. Security properties such as anonymity [4], privacy [20, 6] and strong secrecy [12] are usually formalised by observational equivalence.

We write $\Longrightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}$; we define $\overset{\alpha}{\Longrightarrow}$ to be $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$; we write $\overset{\widehat{\alpha}}{\Longrightarrow}$ for $\overset{\alpha}{\Longrightarrow}$ if α is not τ and $\Longrightarrow$ otherwise. We write $A \Downarrow_a$ when $A \Longrightarrow \nu\widetilde{n}.(\sigma, S, \mathcal{P} \cup \{(\overline{a}\langle M\rangle.P, L)\})$ with $a \notin \widetilde{n}$.

Definition 1. *Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ on pairs of closed extended processes with only private state cells, such that $A\ \mathcal{R}\ B$ implies*

(i) $dom(A) = dom(B)$;
(ii) *if* $A \Downarrow_a$ *then* $B \Downarrow_a$;
(iii) *if* $A \Longrightarrow A'$ *then* $B \Longrightarrow B'$ *and* $A'\ \mathcal{R}\ B'$ *for some* B';
(iv) *for all closing evaluation contexts* $\mathcal{C}$ *with only private cells,* $\mathcal{C}[A]\ \mathcal{R}\ \mathcal{C}[B]$.

Observational equivalence is a contextual equivalence where the contexts model the active attackers who can intercept and forge messages. In the following examples, we illustrate the use of observational equivalence in the stateful language by analysing the untraceability of the RFID tags.

Example 1. We start by analysing a naive protocol for RFID tag identification. The tag simply reads its id and sends it to the reader. We assume the attacker can eavesdrop on the radio frequency signals between the tag and the reader. In other words, all the communications between the tag and the reader are visible to the attacker. The operations on the tag can be modelled by: $P(s) = \texttt{read}\ s\ \texttt{as}\ x.\,\overline{a}\langle x\rangle$. One security concern for RFID tags is to avoid third-party attacker tracking. The attacker is not supposed to trace the tag according to its outputs. Using the definition in [6], the untraceability can be modelled by observational equivalence:

$$(\emptyset, \emptyset, \{(!\,\nu s, id.([s \mapsto id] \mid P(s)), \emptyset)\}) \approx (\emptyset, \emptyset, \{(!\,\nu s, id.([s \mapsto id] \mid !\,P(s)), \emptyset)\})$$

In the left process, each tag s can be used at most once. In the right process, each tag s can be used an unbounded number of times. The above equivalence does not hold, which means this protocol is traceable. By eavesdropping on channel a of the right process, the attacker can get a data sequence: "$id, id, id \cdots$", while a particular *id* can occur at most once in the first process.

Example 2. The OSK protocol [26] is a simple identification protocol for RFID tags which aims to satisfy third-party untraceability. The tag can perform two independent

one-way functions g and h. An initial secret is stored in the tag and is known to the back-end database. On each run of the protocol, the tag computes the hash g of its current value and sends the result to the reader. The reader forwards the message to the back-end database for identification. The tag then updates its value with the hash h of its current value. The operations related to a tag s can be modelled by:

$$T(s) = \texttt{lock}\ s.\texttt{read}\ s\ \texttt{as}\ x.\overline{a}\langle g(x)\rangle.\ s := h(x).\texttt{unlock}\ s$$

Let *RD* be process modelling the reader and back-end database. Similar to Example 1, the untraceability can be represented by

$$\begin{aligned}&(\emptyset, \emptyset, \{(!\,\nu s, k.([s \mapsto k] \mid T(s) \mid RD), \emptyset)\})\\ \approx &(\emptyset, \emptyset, \{(!\,\nu s, k.([s \mapsto k] \mid !T(s) \mid RD), \emptyset)\})\end{aligned}$$

In the second process, for a particular tag s which contains value k, the data sequence observed by the attacker on channel a is "$g(k), g(h(k)), g(h(h(k)))\cdots$". Without knowing the secret k, these appear just random data to the attacker and so the attacker cannot link these data to the same tag. The observational equivalence between these two processes means the attacker cannot identify the multiple runnings of a particular tag. The "$\texttt{lock}\ s\cdots\texttt{unlock}\ s$" ensures exclusive access to the tag. After the reader reads the tag, the tag must be renewed before the next access to the tag; otherwise the tag would be traceable.

The universal quantifier over the contexts makes it difficult to prove observational equivalence. Hence labelled bisimilarity is introduced in [3] to capture observational equivalence. Labelled bisimilarity consists of static equivalence and behavioural equivalence.

Definition 2. *Two processes A and B are* statically equivalent*, written as $A \approx_s B$, if $dom(A) = dom(B)$, and for any terms M and N with $var(M, N) \subseteq dom(A)$, $M\sigma_1 =_\Sigma N\sigma_1$ iff $M\sigma_2 =_\Sigma N\sigma_2$ where $A = \nu\widetilde{n}_1.(\sigma_1, S_1, \mathcal{P}_1)$ and $B = \nu\widetilde{n}_2.(\sigma_2, S_2, \mathcal{P}_2)$ for some $\widetilde{n}_1, \widetilde{n}_2$ such that $(\widetilde{n}_1 \cup \widetilde{n}_2) \cap name(M, N) = \emptyset$.*

Our definition of static equivalence is essentially the same as the one in [3], as the definition in [3] is invariant under structural equivalence already. Although static equivalence is in general undecidable, there are well established ways, including tools, for verifying static equivalence [2, 15, 16, 9, 14]. Static equivalence defines the indistinguishability between the environmental knowledge exposed by two processes. The environmental knowledge is modelled by the substitutions in the extended processes. For example, let $A = \nu k, m.(\{k/x, m/y\}, \emptyset, \emptyset)$ and $B = \nu k.(\{k/x, h(k)/y\}, \emptyset, \emptyset)$. The test $h(x) = y$ fails under the application of A's substitution $\{k/x, m/y\}$, while succeeds under the application of B's substitution $\{k/x, h(k)/y\}$. Hence $A \not\approx_s B$.

Definition 3. *Labelled bisimilarity ($\approx_l$) is the largest symmetric relation $\mathcal{R}$ between pairs of closed extended processes with only private state cells such that $A\ \mathcal{R}\ B$ implies*

1. *$A \approx_s B$;*
2. *if $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq dom(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \stackrel{\widehat{\alpha}}{\Longrightarrow} B'$ such that $A'\ \mathcal{R}\ B'$ for some B'.*

$$\lfloor 0 \rfloor_S = 0 \qquad \lfloor P \mid Q \rfloor_S = \lfloor P \rfloor_S \mid \lfloor Q \rfloor_S \qquad \lfloor \nu n.P \rfloor_S = \nu n.\lfloor P \rfloor_S \text{ if } n \notin \mathcal{N}_s$$
$$\lfloor !\, P \rfloor_S = !\, \lfloor P \rfloor_S \qquad \lfloor u(x).P \rfloor_S = u(x).\lfloor P \rfloor_S \qquad \lfloor \overline{u}\langle M \rangle.P \rfloor_S = \overline{u}\langle M \rangle.\lfloor P \rfloor_S$$
$$\lfloor \texttt{if } M = N \texttt{ then } P \texttt{ else } Q \rfloor_S = \texttt{if } M = N \texttt{ then } \lfloor P \rfloor_S \texttt{ else } \lfloor Q \rfloor_S$$
$$\lfloor s \mapsto M \rfloor_S = \overline{c_s}\langle M \rangle \qquad \lfloor \nu s.P \rfloor_S = \nu c_s.\lfloor P \rfloor_S \text{ if } s \in \mathcal{N}_s$$
$$\lfloor \texttt{lock } s.P \rfloor_S = \begin{cases} c_s(x).\lfloor P \rfloor_{S \cup \{s \mapsto x\}} & \text{if } s \notin dom(S) \text{ and } x \text{ is fresh} \\ 0 & \text{otherwise} \end{cases}$$
$$\lfloor \texttt{unlock } s.P \rfloor_S = \begin{cases} \overline{c_s}\langle M \rangle \mid \lfloor P \rfloor_T & \text{if } S = T \cup \{s \mapsto M\} \\ 0 & \text{otherwise} \end{cases}$$
$$\lfloor \texttt{read } s \texttt{ as } x.P \rfloor_S = \begin{cases} \lfloor P\{M/x\} \rfloor_S & \text{if } S = T \cup \{s \mapsto M\} \\ c_s(x).(\overline{c_s}\langle x \rangle \mid \lfloor P \rfloor_S) & \text{otherwise} \end{cases}$$
$$\lfloor s := M.P \rfloor_S = \begin{cases} \lfloor P \rfloor_{T \cup \{s \mapsto M\}} & \text{if } S = T \cup \{s \mapsto N\} \\ c_s(x).(\overline{c_s}\langle M \rangle \mid \lfloor P \rfloor_S) & \text{otherwise select fresh variable } x \end{cases}$$

Fig. 2. Encoding private state cells with restricted channels

Instead of using arbitrary contexts, labelled bisimilarity relies on the direct comparison of the transitions. The following theorem states that labelled bisimilarity can fully capture observational equivalence:

Theorem 1. *On closed extended processes with only private state cells, it holds that* $\approx = \approx_l$.

3.2 Encoding Private State Cells with Restricted Channels

Private state cells can be encoded by restricted channels. This is an important observation; moreover, we will use this to prove Abadi-Fournet's theorem in the following Section 3.3. However, when modelling security protocols, the drawback of representing private state cells by restricted channels is that it may introduce false attacks when using the automatic tool ProVerif as argued in [7]. The reason is that some features of restricted channels are abstracted away when ProVerif translates process calculus into Horn clauses [13]. To solve this problem, we introduce the primitives for lock, read, write and unlock which will help us design better translations for stateful protocols in ProVerif. This has been demonstrated by the verification of reachability [7], and will be useful in future for verifying observational equivalence.

We encode the extended processes with only private state cells into a subset of the extended processes which do not contain any cell name. Since the target language of the encoding does not have any cell name, we abbreviate extended processes $\nu\widetilde{n}.(\sigma, \emptyset, \{(P_i, \emptyset)\}_{i \in I})$ with no cell name to $\nu\widetilde{n}.(\sigma, \{P_i\}_{i \in I})$.

First we define encoding $\lfloor P \rfloor_S$ in Figure 2 for the plain process P under a given set of state cells $S = \{s_1 \mapsto M_1, \ldots, s_n \mapsto M_n\}$. For each cell s, we select a fresh channel name c_s. The encoding in Figure 2 only affects the part related to cell names, leaving other parts like input and output unchanged. The state cell $s \mapsto M$ and `unlock` s are both encoded by an output $\overline{c_s}\langle M \rangle$ on the restricted channel c_s. The `lock` s is represented by an input $c_s(x)$ on the same channel c_s. To read the cell `read` s `as` x, we

use the input $c_s(x)$ to get the value from the cell and then put the value back $\overline{c_s}\langle x\rangle$, which enables the other operations on cell s in future. To write a new value into the cell $s := N$, we need to first consume the existing $\overline{c_s}\langle M\rangle$ by an input $c_s(x)$ and then generate a new output $\overline{c_s}\langle N\rangle$. Our encoding ensures that there is only one output $\overline{c_s}\langle M\rangle$ available on a specified restricted channel c_s at each moment. When the cell is locked, namely $\overline{c_s}\langle M\rangle$ is consumed by some $c_s(x)$, the other processes that intend to access the cell have to wait until an output $\overline{c_s}\langle N\rangle$ is available.

Let $A = \nu\widetilde{s}, \widetilde{n}.\left(\sigma, \{s_i \mapsto M_i\}_{i\in I}, \{(P_j, L_j)\}_{j\in J}\right)$ be an extended process [1] where $\widetilde{s} \subset \mathcal{N}_s$ and $\widetilde{n} \cap \mathcal{N}_s = \emptyset$. We define the encoding $\lfloor A \rfloor$ as:

$$\lfloor A \rfloor = \nu\widetilde{c_s}, \widetilde{n}.\left(\sigma, \{\overline{c_{s_i}}\langle M_i\rangle\}_{i\in U} \cup \left\{\lfloor P_j \rfloor_{S_j}\right\}_{j\in J}\right)$$

where $U = \{\, i \mid s_i \notin \bigcup_{j\in J} L_j \text{ and } i \in I \,\}$ and $S_j = \{\, s_i \mapsto M_i \mid s_i \in L_j \text{ and } i \in I \,\}$. Intuitively, U is the indices of the unlocked state cells in $\{s_i \mapsto M_i\}_{i\in I}$, and S_j is the set of state cells locked by L_j.

Example 3. Let $A = \nu s.(\emptyset, \{s \mapsto 0\}, \{(T(s), \emptyset)\})$ where $T(s)$ is defined in Example 2. Then $\lfloor A \rfloor = \nu c_s.(\emptyset, \{\overline{c_s}\langle 0\rangle, \lfloor T(s)\rfloor_\emptyset\})$ with $\lfloor T(s)\rfloor_\emptyset = c_s(z).\overline{a}\langle g(z)\rangle.\overline{c_s}\langle h(z)\rangle$ obtained by:

$$\begin{aligned}
\lfloor T(s)\rfloor_\emptyset &= \lfloor \texttt{lock}\ s.\texttt{read}\ s\ \texttt{as}\ x.\overline{a}\langle g(x)\rangle.s := h(x).\texttt{unlock}\ s\rfloor_\emptyset \\
&= c_s(z).\lfloor \texttt{read}\ s\ \texttt{as}\ x.\overline{a}\langle g(x)\rangle.s := h(x).\texttt{unlock}\ s\rfloor_{\{s\mapsto z\}} \\
&= c_s(z).\lfloor \overline{a}\langle g(z)\rangle.s := h(z).\texttt{unlock}\ s\rfloor_{\{s\mapsto z\}} \\
&= c_s(z).\overline{a}\langle g(z)\rangle.\lfloor s := h(z).\texttt{unlock}\ s\rfloor_{\{s\mapsto z\}} \\
&= c_s(z).\overline{a}\langle g(z)\rangle.\lfloor \texttt{unlock}\ s\rfloor_{\{s\mapsto h(z)\}} \\
&= c_s(z).\overline{a}\langle g(z)\rangle.\overline{c_s}\langle h(z)\rangle
\end{aligned}$$

Theorem 2. *For two closed extended processes A, B with only private state cells, we have $A \approx B$ iff $\lfloor A \rfloor \approx^e \lfloor B \rfloor$ where $\approx^e$ is an equivalence defined exactly the same as Definition 1 except the context $\mathcal{C}$ does not contain any cell names.*

3.3 Overview of the Proof of Abadi-Fournet's Theorem

We shall use our Theorem 1 and Theorem 2 to derive Abadi-Fournet's theorem, namely Theorem 1 in [3]. We revise the original applied pi calculus [3] slightly: *active substitutions are only defined on terms of base sort*; otherwise Theorem 1 in [3] does not hold [10].[2] Since the active substitutions in applied pi calculus float everywhere in the extended processes, in order to prove Abadi-Fournet's theorem, we need to normalise

[1] We abbreviate the set $\{\, s_i \mapsto M_i \mid i \in I \,\}$ as $\{s_i \mapsto M_i\}_{i\in I}$.

[2] Here is a counter example: let $A_r = \nu c.(\overline{c}.\overline{a} \mid \{c/x\})$ and $B_r = \nu c.(0 \mid \{c/x\})$. Obviously A_r and B_r are labelled bisimilar since their frames are the same and both have no transitions. However, they are not observationally equivalent. Consider the context $x(y)$, then $A_r \mid x(y) \Downarrow_a$ but $B_r \mid x(y) \not\Downarrow_a$.

the extended processes first. We can transform the extended processes in the applied pi calculus – denoted by A_r, B_r, C_r to avoid confusion – into the extended processes in stateful applied pi calculus by function $\mathcal{T}$ (assume bound names are pairwise-distinct and different from free names): [3]

$$\mathcal{T}(0) = (\emptyset, \emptyset) \qquad \mathcal{T}(\{M/x\}) = (\{M/x\}, \emptyset) \qquad \mathcal{T}(\nu n.A_r) = \nu n.\mathcal{T}(A_r)$$
$$\mathcal{T}(\nu x.A_r) = \nu\widetilde{n}.(\sigma, \mathcal{P}) \text{ if } \mathcal{T}(A_r) = \nu\widetilde{n}.(\sigma \cup \{M/x\}, \mathcal{P})$$
$$\mathcal{T}(A_r^1 \mid A_r^2) = \nu\widetilde{n}_1, \widetilde{n}_2.((\sigma_1 \cup \sigma_2)^*, (\mathcal{P}_1 \cup \mathcal{P}_2)(\sigma_1 \cup \sigma_2)^*)$$
$$\text{if } \mathcal{T}(A_r^i) = \nu\widetilde{n}_i.(\sigma_i, \mathcal{P}_i) \text{ for } i = 1, 2$$
$$\mathcal{T}(A_r) = (\emptyset, \{A_r\}) \text{ in all other cases of } A_r$$

Intuitively, $\mathcal{T}$ pulls out name restrictions, applies active substitutions and separates them from the plain processes, and eliminates variable restrictions. For instance, $\mathcal{T}(\overline{a}\langle x\rangle.\nu n.\overline{a}\langle n\rangle \mid \nu k.\{k/x\}) = \nu k.(\{k/x\}, \{\overline{a}\langle k\rangle.\nu n.\overline{a}\langle n\rangle\})$. This normalisation $\mathcal{T}$ preserves both observational equivalence and labelled bisimilarity:

Theorem 3. *For two closed extended processes A_r and B_r in applied pi calculus,*

1. *A_r and B_r are labelled bisimilar in applied pi iff $\mathcal{T}(A_r) \approx_l \mathcal{T}(B_r)$;*
2. *A_r and B_r are observationally equivalent in applied pi iff $\mathcal{T}(A_r) \approx^e \mathcal{T}(B_r)$;*

With all the theorems ready, now we can prove Abadi-Fournet's theorem:

Corollary 1. *Observational equivalence coincides with labelled bisimilarity in applied pi calculus.*

4 Extending the Language with Public State Cells

4.1 Public State Cells

Hardware modules like TPMs and smart cards are intended to be secure, but an attacker might succeed in finding ways of compromising their tamper-resistant features. Similarly, attackers can potentially hack into databases [1]. We model these attacks by considering that the attacker compromises the private state cells, after which they are public. Protocols may provide some security properties that hold even under such compromises of the hardware or database. A typical example is forward privacy [24] which requires the past events remain secure even if the attacker compromises the device. This will be further discussed in the following Example 8 and Example 9. A cell s not in the scope of νs is public, which enables the attacker to lock the cell, read its contents or overwrite it.

We now give the details of the syntactic additions for public cells and the definition of observational equivalence. To let a private state cell become public, we extend the plain processes in Section 2 with a new primitive `open` $s.P$ Extended processes are defined as before. We extend the transitions in Fig. 1 by a new transition relation $\xrightarrow{\tau(s)}$ defined

[3] We write σ^* for the result of composing the substitution σ with itself repeatedly until an idempotent substitution is reached.

$$\nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(\texttt{read}\ s\ \texttt{as}\ x.P, L)\}) \xrightarrow{\tau(s)} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(P\{M/x\}, L)\})$$
$$\textbf{if } s \notin \tilde{n} \cup L \cup \mathit{locks}(\mathcal{P})$$

$$\nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(s := N.P, L)\}) \xrightarrow{\tau(s)} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto N\}, \mathcal{P} \cup \{(P, L)\})$$
$$\textbf{if } s \notin \tilde{n} \cup L \cup \mathit{locks}(\mathcal{P})$$

$$\nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(\texttt{lock}\ s.P, L)\}) \xrightarrow{\tau(s)} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(P, L \cup \{s\})\})$$
$$\textbf{if } s \notin \tilde{n} \cup L \cup \mathit{locks}(\mathcal{P})$$

$$\nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(\texttt{unlock}\ s.P, L)\}) \xrightarrow{\tau(s)} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(P, L \setminus \{s\})\})$$
$$\textbf{if } s \notin \tilde{n} \cup \mathit{locks}(\mathcal{P}) \text{ and } s \in L$$

$$\nu\tilde{n}, s.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(\texttt{open}\ s.P, L)\}) \xrightarrow{\tau(s)} \nu\tilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P} \cup \{(P, L)\}) \textbf{ if } s \notin \tilde{n}$$

Fig. 3. Internal transitions for public state cells

in Fig. 3 for reasoning about public state cells. These internal transitions specify on which public state cell the operations are performed. The label $\tau(s)$ is necessary when we later define labelled bisimilarity. Note that when a public state cell is locked, we still use the rule $\xrightarrow{\tau}$ defined in Fig. 1 for reading and writing on that cell.

Let $A = \nu\tilde{n}.(\sigma, S, \mathcal{P})$ and we write $\mathit{locks}(A)$ for the set $\mathit{locks}(\mathcal{P}) \setminus \tilde{n}$. We write $\overset{\epsilon}{\Longrightarrow}$ for the reflexive and transitive closure of $\xrightarrow{\tau}$ and $\xrightarrow{\tau(s)}$ for any cell s. We write $A \Downarrow_a$ when $A \overset{\epsilon}{\Longrightarrow} \nu\tilde{n}.(\sigma, S, \mathcal{P} \cup \{(\overline{a}\langle M\rangle.P, L)\})$ with $a \notin \tilde{n}$.

Definition 4. *Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ on pairs of closed extended processes (which may contain public state cells) such that $A\ \mathcal{R}\ B$ implies*

(i) $\mathit{locks}(A) = \mathit{locks}(B)$, $\mathit{fs}(A) = \mathit{fs}(B)$ and $\mathit{dom}(A) = \mathit{dom}(B)$;
(ii) if $A \Downarrow_a$ then $B \Downarrow_a$;
(iii) if $A \overset{\epsilon}{\Longrightarrow} A'$ then $B \overset{\epsilon}{\Longrightarrow} B'$ and $A'\ \mathcal{R}\ B'$ for some B';
(iv) for all closing evaluation contexts $\mathcal{C}$, $\mathcal{C}[A]\ \mathcal{R}\ \mathcal{C}[B]$.

We stick to the original definition of observational equivalence [3] as much as possible in order to capture the intuition of indistinguishability from the attacker's point of view. The definition of observational equivalence on public state cells is similar to the one for private state cells, but the language features of public state cells are significantly different from private state cells. Moreover, the addition of public state cells increases the power of the attacker significantly, as without the name restriction νs for a state cell s, when s is unlocked, the attacker can lock the cell, read its content and overwrite it. To illustrate this point, we start by analysing several examples.

Example 4. The attacker can lock the unlocked public state cells. Assume

$$A = (\emptyset, \{s \mapsto 0\}, \{(\overline{c}\langle b\rangle, \emptyset)\})$$
$$B = (\emptyset, \{s \mapsto 0\}, \{(\texttt{read}\ s\ \texttt{as}\ x.\, \overline{c}\langle b\rangle, \emptyset)\})$$

A and B are not observationally equivalent. Let $\mathcal{C} = (\text{-}, \text{-}, \{(0, \{s\})\}\ \text{-})$. The context $\mathcal{C}$ does nothing but holds the lock on cell s and it will never release the lock. So we have $\mathcal{C}[A] \Downarrow_c$ but $\mathcal{C}[B] \not\Downarrow_c$ because reading cell s in B is blocked forever by context $\mathcal{C}$.

Example 5. The attacker can read an unlocked public state cell. Assume

$$A = (\emptyset, \{s \mapsto 0\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset)\})$$
$$B = (\emptyset, \{s \mapsto 1\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset)\})$$

Cell s is unlocked in both A and B. Both A and B can write 0 or 1 to the cell s arbitrary number of times. The only difference between A and B is the initial values in cell s. A and B are not observationally equivalent because the context

$$\mathcal{C} = (\text{-}, \text{-}, \{(\texttt{read}\ s\ \texttt{as}\ x.\ \texttt{if}\ x = 0\ \texttt{then}\ \overline{c}\langle b\rangle, \{s\})\}\ \text{-})$$

can distinguish them. The context $\mathcal{C}$ holds the lock of cell s, thus no one can change the value in s when $\mathcal{C}$ reads the value. We have $\mathcal{C}[A] \Downarrow_c$ but $\mathcal{C}[B] \not\Downarrow_c$.

In comparison, the following processes are observationally equivalent:

$$A' = (\emptyset, \{s \mapsto 0\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset), (\texttt{unlock}\ s, \{s\})\})$$
$$B' = (\emptyset, \{s \mapsto 1\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset), (\texttt{unlock}\ s, \{s\})\})$$

Cell s is locked in both A' and B'. When a cell is locked, the attacker cannot see its value until it is unlocked. Both A' and B' can adjust the value of cell s after `unlock` s. Assume

$$A' \xrightarrow{\tau(s)} (\emptyset, \{s \mapsto 0\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset), (0, \emptyset)\})$$

Then B' can match this transition by first unlocking the cell s and then doing a writing $s := 0$ and evolving to exactly the same process:

$$B' \xrightarrow{\tau(s)} (\emptyset, \{s \mapsto 1\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset), (0, \emptyset)\})$$
$$\xRightarrow{\tau(s)} (\emptyset, \{s \mapsto 0\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset), (0, \emptyset)\})$$

Intuitively, the locked or unlocked status of a public state cell is observable by the environment. Therefore, we require $\mathit{locks}(A) = \mathit{locks}(B)$ and $\mathit{fs}(A) = \mathit{fs}(B)$ in the definition of observational equivalence. Furthermore, without this condition, this definition would not yield an equivalence relation, as transitivity does not hold in general. For example, consider the following extended processes,

$$A = (\emptyset, \{s \mapsto 0\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset), (!\,\texttt{lock}\ s.\texttt{unlock}\ s, \emptyset)\})$$
$$B = (\emptyset, \{s \mapsto 1\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset), (!\,\texttt{lock}\ s.\texttt{unlock}\ s, \emptyset), (\texttt{unlock}\ s, \{s\})\})$$
$$C = (\emptyset, \{s \mapsto 1\}, \{(!\,s := 0, \emptyset), (!\,s := 1, \emptyset), (!\,\texttt{lock}\ s.\texttt{unlock}\ s, \emptyset)\})$$

Without the condition, then A and B would be equivalent, as well as B and C, because the value in s can always be adjusted to be exactly the same after `unlock` s. But A and C are not equivalent as analysed in Example 5.

Example 6. The value in an unlocked public state cell is a part of the attacker's knowledge. Assume

$$A = \nu k.(\emptyset, \{s \mapsto k\}, \{(s := 0.a(x).\texttt{if}\ x = k\ \texttt{then}\ \overline{c}\langle b\rangle, \emptyset)\})$$
$$B = \nu k.(\emptyset, \{s \mapsto k\}, \{(s := 0.a(x), \emptyset)\})$$

A and B are not observationally equivalent. Let $\mathcal{C} = (\text{-}, \text{-}, \{(\texttt{read}\ s\ \texttt{as}\ y.\ \overline{a}\langle y\rangle, \emptyset)\}\ \text{-})$. Then $\mathcal{C}[A] \Downarrow_c$ but $\mathcal{C}[B] \not\Downarrow_c$ because

$$\begin{aligned}\mathcal{C}[A] &\xrightarrow{\tau(s)} \nu k.\,(\emptyset, \{s \mapsto k\}, \{(\overline{a}\langle k\rangle, \emptyset), (s := 0.a(x).\texttt{if}\ x = k\ \texttt{then}\ \overline{c}\langle b\rangle, \emptyset)\}) \\ &\xrightarrow{\tau(s)} \nu k.\,(\emptyset, \{s \mapsto 0\}, \{(\overline{a}\langle k\rangle, \emptyset), (a(x).\texttt{if}\ x = k\ \texttt{then}\ \overline{c}\langle b\rangle, \emptyset)\}) \\ &\Longrightarrow\ \ \nu k.\,(\emptyset, \{s \mapsto 0\}, \{(\overline{c}\langle b\rangle, \emptyset)\})\end{aligned}$$

But there is no output on channel c in $C[B]$. Hence $A \not\approx B$.

Example 7. The attacker can write an arbitrary value into an unlocked public cell. Assume two extended processes

$$\begin{aligned}A &= (\emptyset, \{s \mapsto 0\}, \{(s := 0.\, s := 0, \emptyset)\}) \\ B &= (\emptyset, \{s \mapsto 0\}, \{(s := 0, \emptyset)\})\end{aligned}$$

A and B are not observationally equivalent. Applying $\mathcal{C} = (\text{-}, \text{-}, \{(s := 1.s := 1, \emptyset)\}\ \text{-})$ to both A and B, the interleaving of $s := 0$ and $s := 1$ can generate a sequence of values $0, 1, 0, 1, 0$ in cell s in $\mathcal{C}[A]$, while the closest sequence generated by $\mathcal{C}[B]$ should be $0, 1, 0, 1, 1$. So when the attacker keeps on reading the value in cell s, he would be able to notice the difference.

Instead of using the primitive $\texttt{open}\ \ s$, an alternative way for making a private state cell become public is to send cell name s on a free channel $\overline{c}\langle s\rangle.P$. The reason we choose the primitive $\texttt{open}\ \ s.P$ here is because sending and receiving cell names through channels is too powerful, and will lead to soundness problems when we define labelled bisimilarity later. For example, let

$$\begin{aligned}A &= (\emptyset, \emptyset, \{(c(x).\texttt{read}\ x\ \texttt{as}\ z.\overline{a}\langle z\rangle, \emptyset)\}) \\ B &= (\emptyset, \emptyset, \{(c(x), \emptyset)\})\end{aligned}$$

In the presence of input and output for cell names, A and B are not observationally equivalent. Let $\mathcal{C} = (\text{-}, \{t \mapsto 0\}\ \text{-}, \{(\overline{c}\langle t\rangle, \emptyset)\}\ \text{-})$. The context $\mathcal{C}$ brings his own state cell $t \mapsto 0$ and we have $\mathcal{C}[A] \Downarrow_a$ but $\mathcal{C}[B] \not\Downarrow_a$. That is to say, in order to define a sound labelled bisimilarity, we have to allow a process like $(\emptyset, \emptyset, \{(\texttt{read}\ t\ \texttt{as}\ z.\ \overline{a}\langle z\rangle, \emptyset)\})$ to perform the reading even without a state cell $t \mapsto 0$. This requires a rather complex definition of labelled bisimilarity, while what we want is to simply free a cell which can be achieved by $\texttt{open}\ \ s.P$.

Now we give examples of the use of public state cells for modelling protocols and security properties. Another security concern for RFID tags is forward privacy [26]. In the following Example 8 and Example 9, we shall illustrate how to model forward privacy by public state cells. Forward privacy requires that even the attacker breaks the tag, the past events should still be untraceable. Public state cells enable us to model the compromised tags.

Example 8. We consider an improved version of the naive protocol in Example 1. Instead of simply outputting the tag's *id*, the tag generates a random number r, hashes its

id concatenated with r and then sends both r and $h(id, r)$ to the reader for identification. This can be modelled by:

$$Q(s) = \texttt{read}\ s\ \texttt{as}\ x.\, \nu r.\, \overline{a}\langle (r, h(x, r))\rangle$$

Upon receiving the value, the reader identifies the tag by performing a brute-force search of its known ids. By observing on channel a, the attacker can get the data pairs from a particular tag s: $(r_1, h(id, r_1)), (r_2, h(id, r_2)), (r_3, h(id, r_3)) \cdots$. Since the hash function is not invertible, without knowing the value of *id*, these data appear as just random data to the attacker. Hence this improved version satisfies the untraceability defined in Example 1. But it does not have the forward privacy. Let *RD* be process modelling the reader and back-end database. The forward privacy can be characterised by the observational equivalence

$$\begin{aligned}&(\emptyset, \emptyset, \{(\,!\,\nu s, id.([s \mapsto id] \mid Q(s) \mid \texttt{open}\ s.\,!\,Q(s) \mid RD), \emptyset)\})\\ &\quad\approx\ (\emptyset, \emptyset, \{(\,!\,\nu s, id.([s \mapsto id] \mid\,!\,Q(s) \mid \texttt{open}\ s \mid RD), \emptyset)\})\end{aligned}$$

The primitive $\texttt{open}\ s$ makes the private state cell s become public. Before the cell s is broken, the attacker cannot decide how the system runs. In other words, whether the tag s is used for only once, namely $Q(s)$, or is used for arbitrary number of times, namely $!\,Q(s)$, it is out of the control of the attacker. But after the tag is broken, the attacker fully controls the tag, so he knows when and where the tag is used. Despite knowing the events that happen after the tag is broken, the attacker should still not be able to trace the past events. Therefore, in the first process, we add $!\,Q(s)$ after $\texttt{open}\ s$ to model this scenario. Intuitively, only the events before the tag is broken may be different while the events after the tag is broken are exactly the same. Hence the above observational equivalence can capture forward privacy.

However the above equivalence does not hold which means there is no forward privacy in this protocol. The attacker can obtain the *id* from the broken tag and then verify whether the previously gathered data $(r_1, h(id, r_1))$ and $(r_2, h(id, r_2))$ refer to the same tag *id* by hashing *id* with r_1 (or r_2) and then comparing the result with $h(id, r_1)$ (or $h(id, r_2)$).

Example 9. Continuing with the OSK protocol in Example 2, we model the forward privacy by the observational equivalence:

$$\begin{aligned}&(\emptyset, \emptyset, \{(\,!\,\nu s, k.([s \mapsto k] \mid T(s) \mid \texttt{open}\ s.\,!\,T(s) \mid RD), \emptyset)\})\\ &\quad\approx\ (\emptyset, \emptyset, \{(\,!\,\nu s, k.([s \mapsto k] \mid\,!\,T(s) \mid \texttt{open}\ s \mid RD), \emptyset)\})\end{aligned}$$

Before the tag is broken, the attacker can obtain the data sequence $g(k), g(h(k))$, $g(h(h(k))) \cdots$ by eavesdropping on channel a. Right after each reading, the value in the tag will be updated to the hash of previous value: $h(k), h(h(k)), h(h(h(k))) \cdots$. When the tag is broken, the attacker will get from the tag a value $h^i(k)$ for some integer i. This value is not helpful for the attacker to infer whether the data $g(k), g(h(k)), \cdots$, $g(h^{i-1}(k))$ are from the same tag. Hence the OSK protocol can ensure the forward privacy.

In order to ease the verification of observational equivalence which is defined using the universal quantifier over contexts, we shall define labelled bisimilarity which replaces quantification over contexts by suitably labelled transitions. The traditional definition for labelled bisimilarity is neither sound nor complete w.r.t. observational equivalence in the presence of public state cells. We propose a novel definition for labelled bisimilarity and show how it solves all the problems caused by public state cells.

For a given cell s, we define $\stackrel{\tau(s)}{\Longrightarrow}$ to be the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$ and $\stackrel{\tau(s)}{\longrightarrow}$. We still use α to range over $\tau, a(M), \overline{a}\langle c\rangle, \nu c.\overline{a}\langle c\rangle$ and $\nu x.\overline{a}\langle x\rangle$, and use $\Longrightarrow$ for the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$, and use $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ for $\stackrel{\alpha}{\Longrightarrow}$ if α is not τ and $\Longrightarrow$ otherwise.

To define labelled bisimilarity, we need an auxiliary transition relation $\stackrel{s:=N}{\longrightarrow}$ for setting the values of public state cells:

$$\nu\widetilde{n}.(\sigma, S \cup \{s \mapsto M\}, \mathcal{P}) \stackrel{s:=N}{\longrightarrow} \nu\widetilde{n}.(\sigma, S \cup \{s \mapsto N\sigma\}, \mathcal{P}) \quad \textbf{if } s \notin \widetilde{n} \cup \mathit{locks}(\mathcal{P}) \text{ and } \mathit{name}(N) \cap \widetilde{n} = \emptyset$$
$$\nu\widetilde{n}.(\sigma, S, \mathcal{P}) \stackrel{s:=N}{\longrightarrow} \nu\widetilde{n}.(\sigma, S, \mathcal{P}) \quad \textbf{if } s \in \widetilde{n} \cup \mathit{locks}(\mathcal{P})$$

The first rule of $\stackrel{s:=N}{\longrightarrow}$ represents the attacker's ability to overwrite the public state cells. The second rule does not change the value of the cell s and is just for compatibility with `unlock` s and `open` s in Definition 5. We write $A \stackrel{s:=N}{\longrightarrow}\stackrel{\tau(s)}{\Longrightarrow} A'$ for the combination of transitions $A \stackrel{s:=N}{\longrightarrow} B$ and $B \stackrel{\tau(s)}{\Longrightarrow} A'$ for some B.

Definition 5. *Labelled bisimilarity ($\approx_l$) is the largest symmetric relation $\mathcal{R}$ between pairs of closed extended processes $A_i = \nu\widetilde{n}_i.(\sigma_i, S_i, \mathcal{P}_i)$ with $i = 1, 2$ such that $A_1 \mathcal{R} A_2$ implies*

1. *$\mathit{locks}(A_1) = \mathit{locks}(A_2)$, $\mathit{fs}(A_1) = \mathit{fs}(A_2)$ and $\mathit{dom}(A_1) = \mathit{dom}(A_2)$;*
2. *Let U be the set of unlocked public state cells whose value is not already given in the substitutions of A_1 and A_2, that is*

$$U = \{\, s \mid s \in \mathit{fs}(A_1) \setminus \mathit{locks}(A_1), \nexists x \in \mathit{dom}(\sigma_1) \text{ s.t. } S_1(s) = x\sigma_1 \text{ and } S_2(s) = x\sigma_2 \,\}$$

Select a fresh base variable x_s for each $s \in U$. Let

$$A_i^e = \nu\widetilde{n}_i.(\sigma_i \cup \{S_i(s)/x_s\}_{s \in U}, S_i, \mathcal{P}_i) \text{ for } i = 1, 2$$

Then

(a) $A_1^e \approx_s A_2^e$;

(b) if $A_1^e \stackrel{s:=N}{\longrightarrow}\stackrel{\tau(s)}{\longrightarrow} B_1$ with $\mathit{var}(N) \subseteq \mathit{dom}(A_1^e)$, then there exists B_2 such that $A_2^e \stackrel{s:=N}{\longrightarrow}\stackrel{\tau(s)}{\Longrightarrow} B_2$ and $B_1 \mathcal{R} B_2$;

(c) if $A_1^e \stackrel{\alpha}{\longrightarrow} B_1$ and $\mathit{fv}(\alpha) \subseteq \mathit{dom}(A_1^e)$ and $\mathit{bnv}(\alpha) \cap \mathit{fnv}(A_2^e) = \emptyset$, then there exists B_2 such that $A_2^e \stackrel{\widehat{\alpha}}{\Longrightarrow} B_2$ and $B_1 \mathcal{R} B_2$.

The static equivalence $A_1^e \approx_s A_2^e$ in Definition 5 is exactly the same as the one defined in Definition 2. Before we compare the static equivalence and the transitions in labelled bisimilarity, we extend A_i to A_i^e with values from unlocked public state cells. This is to reflect the fact that attacker's ability to read values from these cells.

Example 10. Consider the extended processes A and B in Example 5. As we have already shown, A and B are not observationally equivalent. Hence they are not supposed to be labelled bisimilar. We first extend A and B to A^e and B^e respectively:

$$A^e = (\{0/z\}, \{s \mapsto 0\}, \{(!\, s := 0, \emptyset), (!\, s := 1, \emptyset)\})$$
$$B^e = (\{1/z\}, \{s \mapsto 1\}, \{(!\, s := 0, \emptyset), (!\, s := 1, \emptyset)\})$$

Clearly the static equivalence between A^e and B^e does not hold, namely $A^e \not\approx_s B^e$, because the test $z = 0$ can distinguish them. Thus we have $A \not\approx_l B$.

The extension is not only for comparing the static equivalence, but also for comparing the transitions. In labelled bisimilarity, we compare the transitions starting from the extensions A^e and B^e, rather than the original processes A and B. The reason is that we need to keep a copy of the cell values, otherwise we would lose the values when someone overwrites the cells.

Example 11. Consider the extended processes A and B in Example 6. The extension A^e of A can perform the following transition:

$$\begin{aligned} A^e = {} & \nu k.(\{k/z\}, \{s \mapsto k\}, \{(s := 0.a(x).\texttt{if } x = k \texttt{ then } \overline{c}\langle b\rangle, \emptyset)\}) \\ \xrightarrow{\tau(s)} {} & \nu k.(\{k/z\}, \{s \mapsto 0\}, \{(a(x).\texttt{if } x = k \texttt{ then } \overline{c}\langle b\rangle, \emptyset)\}) \\ \overset{a(z)}{\Longrightarrow} {} & \nu k.(\{k/z\}, \{s \mapsto 0\}, \{(\overline{c}\langle b\rangle, \emptyset)\}) \\ \xrightarrow{\overline{c}\langle b\rangle} {} & \nu k.(\{k/z\}, \{s \mapsto 0\}, \{(0, \emptyset)\}) \end{aligned}$$

But it is impossible for B's extension $B^e = \nu k.(\{k/z\}, \{s \mapsto k\}, \{(s := 0.\ a(x), \emptyset)\})$ to perform an output on channel c. Hence $A \not\approx_l B$.

We use $\xrightarrow{s:=N}\xrightarrow{\tau(s)}$ rather than $\xrightarrow{\tau(s)}$ in labelled bisimilarity because the attacker can set any unlocked public state cell to an arbitrary value. We shall illustrate this point by the following two examples.

Example 12. Assume

$$A = (\{0/y, 1/z\}, \{s \mapsto 0\}, \{(\texttt{read } s \texttt{ as } x.\ \texttt{if } x = 1 \texttt{ then } \overline{c}\langle 0\rangle, \emptyset)\})$$
$$B = (\{0/y, 1/z\}, \{s \mapsto 0\}, \emptyset)$$

A and B are not observationally equivalent. Applying context $\mathcal{C} = (\emptyset, \emptyset, \{(s := 1, \emptyset)\})$ to A and B, we can see that $\mathcal{C}[A] \Downarrow_c$ but $\mathcal{C}[B] \not\Downarrow_c$.

Now we shall distinguish them in labelled bisimilarity. Since the current value in cell s is 0 which has already been stored in variable y, we don't need to extend A and B.

Then A can perform the following transition

$$A \xrightarrow{s:=1} \xrightarrow{\tau(s)} (\{0/y, 1/z\}, \{s \mapsto 1\}, \{(\texttt{if } 1 = 1 \texttt{ then } \overline{c}\langle a\rangle, \emptyset)\}) \xrightarrow{\overline{c}\langle a\rangle} (\{0/y, 1/z\}, \{s \mapsto 1\}, \{0, \emptyset\})$$

But there is no way for B to perform an output action. Hence $A \not\approx_l B$.

Example 13. As illustrated in Example 7, A and B are not observationally equivalent. In labelled bisimilarity, we extend A and perform the transitions $\xrightarrow{s:=1}\xrightarrow{\tau(s)}$ twice, then we will reach a process $A' = (\{0/x, 0/z\}, \{s \mapsto 0\}, \{(0, \emptyset)\})$, while the best B can do to match A is to reach a process $B' = (\{0/x, 1/z\}, \{s \mapsto 0\}, \{(0, \emptyset)\})$ and $A' \not\approx_s B'$.

Note that the transition $\xrightarrow{s:=N}$ is not included in $\xrightarrow{\alpha}$. We only need to use $\xrightarrow{s:=N}$ to change the value of the unlocked public state cell s when the processes perform some actions related to s. Comparing the combination of two transitions together ($\xrightarrow{s:=N}\xrightarrow{\tau(s)}$) in Definition 5 optimises the definition to be better suited as an assisted tool for analysing observational equivalence. Otherwise, if we follow the traditional way to define labelled bisimilarity, i.e. comparing $A_1^e \xrightarrow{s:=N} B_1^e$ and $A_1^e \xrightarrow{\tau(s)} B_1^e$ separately, the action $\xrightarrow{s:=N}$ would generate infinitely many unnecessary branches. For example, let $A = (\emptyset, \{s \mapsto 0\}, \emptyset)$. Even there is no action, A could keep on performing $\xrightarrow{s:=N}$ and would never stop: $A \xrightarrow{s:=1} (\emptyset, \{s \mapsto 1\}, \emptyset) \xrightarrow{s:=2} (\emptyset, \{s \mapsto 2\}, \emptyset) \xrightarrow{s:=3} (\emptyset, \{s \mapsto 3\}, \emptyset) \cdots$

We require $A_1^e \xrightarrow{s:=N}\xrightarrow{\tau(s)} B_1$ to be matched by $A_2^e \xrightarrow{s:=N}\xRightarrow{\tau(s)} B_2$ with the same s in the action in labelled bisimilarity. In other words, A_2^e can only match the transition of A_1^e by at most operating on the same cell s. This is equal to say the attacker holds the locks of all the unlocked public cell except cell s in A_1^e. If A_1^e does not do act on cell s, then A_2^e are not allowed to match A_1^e by operating on s.

Example 14. Extend A and B in Example 4 to $A^e = (\{0/z\}, \{s \mapsto 0\}, \{(\overline{c}\langle b\rangle, \emptyset)\})$ and $B^e = (\{0/z\}, \{s \mapsto 0\}, \{(\texttt{read } s \texttt{ as } x.\, \overline{c}\langle b\rangle, \emptyset)\})$. We can see that $A^e \xrightarrow{\overline{c}\langle b\rangle} (\emptyset, \{s \mapsto 0\}, \{(0, \emptyset)\})$, but there is no way for B^e to do the same output action $\overline{c}\langle b\rangle$ without going through the reading on cell s. Hence $A \not\approx_l B$.

In the presence of public state cells, labelled bisimilarity is both sound and complete with respect to observational equivalence.

Theorem 4. *In the presence of public state cells,* $\approx_l = \approx$.

5 Conclusion

We present a stateful language which is a general extension of applied pi calculus with state cells. We stick to the original definition of observational equivalence [3] as much as possible to capture the intuition of indistinguishability from the attacker's point of view, while design the labelled bisimilarity to furthest abstract observational equivalence. When all the state cells are private, we prove that observational equivalence coincides with labelled bisimilarity, which implies Abadi-Fournet's theorem in a revised

version of applied pi calculus. In the presence of public state cells, we devise a labelled bisimilarity which is proved to coincide with observational equivalence. In future, we plan to develop a compiler for bi-processes with state cells to automatically verify the observational equivalence, extending the techniques of ProVerif.

References

[1] LinkedIn investigates hacking claims, http://www.guardian.co.uk/technology/2012/jun/06/linkedin-hacking
[2] Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. Theor. Comput. Sci. 367(1-2), 2–32 (2006)
[3] Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proc. 28th Symposium on Principles of Programming Languages (POPL 2001), pp. 104–115. ACM Press (2001)
[4] Abadi, M., Fournet, C.: Private authentication. Theor. Comput. Sci. 322(3), 427–476 (2004)
[5] Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. In: 4th ACM Conference on Computer and Communications Security, pp. 36–47. ACM Press (1997)
[6] Arapinis, M., Chothia, T., Ritter, E., Ryan, M.: Analysing unlinkability and anonymity using the applied pi calculus. In: Proceedings of IEEE 23rd Computer Security Foundations Symposium, CSF 2010, pp. 107–121 (2010)
[7] Arapinis, M., Ritter, E., Ryan, M.D.: Statverif: Verification of stateful processes. In: Proceedings of IEEE 24th Computer Security Foundations Symposium, CSF 2011, pp. 33–47 (2011), http://markryan.eu/research/statverif/
[8] Backes, M., Maffei, M., Unruh, D.: Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In: IEEE Symposium on Security and Privacy, pp. 202–215 (2008)
[9] Baudet, M., Cortier, V., Delaune, S.: YAPA: A generic tool for computing intruder knowledge. ACM Transactions on Computational Logic 14 (2013)
[10] Bengtson, J., Johansson, M., Parrow, J., Victor, B.: Psi-calculi: a framework for mobile processes with nominal data and logic. Logical Methods in Computer Science 7(1) (2011)
[11] Bhargavan, K., Fournet, C., Corin, R., Zalinescu, E.: Cryptographically verified implementations for TLS. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS 2008, pp. 459–468. ACM (2008)
[12] Blanchet, B.: Automatic Proof of Strong Secrecy for Security Protocols. In: IEEE Symposium on Security and Privacy, pp. 86–100 (2004)
[13] Blanchet, B.: Automatic Verification of Correspondences for Security Protocols. Journal of Computer Security 17(4), 363–434 (2009)
[14] Chadha, R., Ciobâcă, Ş., Kremer, S.: Automated verification of equivalence properties of cryptographic protocols. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 108–127. Springer, Heidelberg (2012)
[15] Cheval, V., Comon-Lundh, H., Delaune, S.: Automating security analysis: Symbolic equivalence of constraint systems. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 412–426. Springer, Heidelberg (2010)
[16] Cheval, V., Comon-Lundh, H., Delaune, S.: Trace equivalence decision: Negative tests and non-determinism. In: Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS 2011), pp. 321–330 (2011)
[17] Cheval, V., Cortier, V., Delaune, S.: Deciding equivalence-based properties using constraint solving. Theoretical Computer Science 492, 1–39 (2013)

[18] Cortier, V., Delaune, S.: A method for proving observational equivalence. In: CSF 2009, pp. 266–276. IEEE Computer Society Press (July 2009)
[19] Cortier, V., Rusinowitch, M., Zalinescu, E.: Relating two standard notions of secrecy. Logical Methods in Computer Science 3, 1–29 (2007)
[20] Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security (2009)
[21] Delaune, S., Kremer, S., Ryan, M.D.: Symbolic bisimulation for the applied pi calculus. Journal of Computer Security 18(2), 317–377 (2010)
[22] Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: Formal analysis of protocols based on TPM state registers. In: Proc. of the 24th IEEE Computer Security Foundations Symposium (CSF 2011). IEEE Computer Society Press (2011)
[23] Dreyer, D., Neis, G., Rossberg, A., Birkedal, L.: A relational modal logic for higher-order stateful ADTs. In: Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, pp. 185–198 (2010)
[24] Garcia, F.D., van Rossum, P.: Modeling privacy for off-line RFID systems. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 194–208. Springer, Heidelberg (2010)
[25] Guttman, J.D.: Fair exchange in strand spaces. Journal of Automated Reasoning (2012)
[26] Koutarou, M.O., Suzuki, K., Kinoshita, S.: Cryptographic approach to "privacy-friendly" tags. RFID Privacy Workshop (2003)
[27] Künnemann, R., Steel, G.: YubiSecure? formal security analysis results for the Yubikey and YubiHSM. In: Jøsang, A., Samarati, P., Petrocchi, M. (eds.) STM 2012. LNCS, vol. 7783, pp. 257–272. Springer, Heidelberg (2013)
[28] Liu, J.: A proof of coincidence of labeled bisimilarity and observational equivalence in applied pi calculus. Technical Report, ISCAS-SKLCS-11-05 (2011), `http://mail.ios.ac.cn/~jliu/papers/Proof.pdf`
[29] Liu, J., Lin, H.: A complete symbolic bisimulation for full applied pi calculus. Theoretical Computer Science 458, 76–112 (2012)
[30] Mödersheim, S.: Abstraction by set-membership: verifying security protocols and web services with databases. In: Proc. 17th ACM Conference on Computer and Communications Security (CCS 2010), pp. 351–360. ACM (2010)
[31] Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, B.: The Modelling and Analysis of Security Protocols. Pearson Education (2001)
[32] Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
[33] Schmidt, B., Meier, S., Cremers, C., Basin, D.: Automated analysis of diffie-hellman protocols and advanced security properties. In: 25th Computer Security Foundations Symposium (CSF), pp. 78–94. IEEE (2012)
[34] Turon, A., Dreyer, D., Birkedal, L.: Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency. In: ICFP 2013, pp. 377–390 (2013)