Automatic verification of privacy properties in the applied pi calculus[⋆]

Stéphanie Delaune, Mark Ryan, and Ben Smyth

Abstract We develop a formal method verification technique for cryptographic protocols. We focus on proving observational equivalences of the kind $P \sim Q$, where the processes P and Q have the same structure and differ only in the choice of terms. The calculus of ProVerif, a variant of the applied pi calculus, makes some progress in this direction. We expand the scope of ProVerif, to provide reasoning about further equivalences. We also provide an extension which allows modelling of protocols which require global synchronisation. Finally we develop an algorithm to enable automated reasoning. We demonstrate the practicality of our work with two case studies.

1 Introduction

Security protocols are small distributed programs that aim to provide some security related objective over a public communications network like the Internet. Considering the increasing size of networks and their dependence on cryptographic protocols, a high level of assurance is needed in the correctness of such protocols. It is difficult to ascertain whether or not a cryptographic protocol satisfies its security requirements. Numerous protocols have appeared in literature and have subsequently been found to be flawed [13, 14, 5]. Typically, cryptographic protocols are expected to achieve their objectives in

Stéphanie Delaune
LSV, ENS Cachan & CNRS & INRIA, France, e-mail: delaune@lsv.ens-cachan.fr

Mark Ryan · Ben Smyth
School of Computer Science, University of Birmingham, UK, e-mail: {B.A.Smyth, M.D.Ryan}@cs.bham.ac.uk

⋆ This work has been partly supported by the ARA SESUR project AVOTÉ and the EPSRC projects *Verifying anonymity and privacy properties* (EP/E040829/1) & *UbiVal* (EP/D076625/1).

the presence of an attacker that is assumed to have full control of the network (sometimes called the Dolev-Yao attacker). He can eavesdrop, replay, inject and block messages. The attacker can also modify them by performing cryptographic operations when in possession of the required keys. Furthermore the attacker may be in control of one or more of the protocol's participants. With no more than the abilities listed, and irrespective of the underlying cryptographic algorithms, numerous protocols have been found to be vulnerable to attack. Formal verification of cryptographic protocols is therefore required to ensure that cryptographic protocols can be deployed without the risk of damage.

Traditionally cryptographic protocols have been required to satisfy secrecy and authentication properties [6]. These requirements have been successfully verified by modelling them as reachability problems. Current research into applications such as electronic voting, fair exchange and trusted computing has resulted in a plethora of new requirements which protocols must satisfy (e.g. [11, 4]). Some of these properties cannot easily be expressed using traditional reachability techniques but can be written as equivalences. For example, the privacy, receipt-freeness and coercion-resistance properties of electronic voting protocols can be expressed using equivalences (see [12, 7]).

We focus on proving equivalences of the kind $P \sim Q$, where the processes P and Q have the same structure and differ only in the choice of terms. For example, the secret ballot (privacy) property of an electronic voting protocol can be expressed as

$$P(skva, v_1) \mid P(skvb, v_2) \; \sim \; P(skva, v_2) \mid P(skvb, v_1)$$

where P is the voter process with two parameters: its secret key ($skva$, $skvb$) and the candidate for whom he wish to cast their vote (here v_1, v_2). Historically many applications of equivalences to prove security requirements of cryptographic protocols have relied upon hand written proofs [12, 7]. Such proofs are time consuming and error prone. Accordingly, we direct our attention to automated techniques. The calculus developed by Blanchet *et al.* makes some progress in this direction [3]. However, the method developed for proving observational equivalence is not complete and is unable to prove certain interesting equivalences.

Contribution. We build upon [3] to provide reasoning about further equivalences (see Section 2). We also extend the syntax to allow the modelling of a new class of processes which require global synchronisation. Finally we develop an algorithm to enable automated reasoning about security requirements. The focus of our work is to model the privacy properties increasingly found in cryptographic protocols (Section 3). We demonstrate the practical application of our contribution with case studies (Sections 4 and 5). Using our approach we provide the first automated proof that the electronic voting protocol due to Fujioka, Okamoto & Ohta (FOO) [10] satisfies privacy. As a second case study we provide a formal methods proof that the Direct Anonymous Attestation (DAA) [4] protocol also satisfies privacy (the DAA authors provided a provable security proof). An extended version of this paper [9] and our ProVerif source code are available at http://www.cs.bham.ac.uk/~bas/.

Related work. Kremer & Ryan [12] have previously demonstrated the electronic voting protocol FOO satisfies fairness, eligibility and privacy. The first two properties were verified automatically using ProVerif, and the third relied on a hand proof. Backes *et al.* [2] model a variant of DAA and provide some proofs. We observe that their model is not accurate with regards to DAA due to some subtleties in their formalisation. Nevertheless their idea of modelling synchronisation by private channel communication influenced the design of our translator.

2 Calculus of ProVerif

The process calculi of Blanchet *et al.* [3], used by the tool ProVerif, is a variant of the applied pi calculus [1], a process calculi for formally modelling concurrent systems and their interactions. In this paper we use the phrase *calculus of ProVerif* to mean the calculus defined in [3], and *ProVerif software tool* to refer to the software tool developed in accompaniment of [3].

2.1 Syntax and informal semantics

The calculus assumes an infinite set of *names* and an infinite set of *variables*. It also assumes a *signature* Σ, i.e. a finite set of *function symbols* each with an associated arity. A function symbol with arity 0 is also called a *constant*. We distinguish two categories of function symbols: *constructors* f and *destructors* g and we use h to range over both. We use standard notation for function application, i.e. $h(M_1, \ldots, M_n)$. Destructors are partial, non-deterministic operations, that processes can apply to terms. They represent primitives that can visibly succeed or fail, while constructors and the associated equational theory apply to primitives that always succeed but may return "junk". The grammar for terms/term evaluations is given below.

$M, N ::=$	term	$D ::=$	term evaluation
a, b, c	name	M	term
x, y, z	variable	$\text{choice}[D, D']$	choice term eval.
$\text{choice}[M, M']$	choice term	$h(D_1, \ldots, D_n)$	function eval.
$f(M_1, \ldots, M_n)$	constructor		

We equip the signature Σ with an *equational theory*, say E, i.e. a finite set of equations of the form $M_i = N_i$, where M_i and N_i are terms without names. The equational theory is then obtained from this set of equations by reflexive, symmetric and transitive closure, closure by substitution of terms for variables and closure by context application. We write $M =_{\mathsf{E}} N$ (resp. $M \neq_{\mathsf{E}} N$) for equality (resp. inequality) modulo E.

Processes are built up in a similar way to processes in the pi calculus, except that messages can contain terms/term evaluations (rather than just names). In the grammar described below, M and N are terms, D is a term evaluation, a is a name, x a variable and t an integer.

$P, Q, R ::=$	processes
$\quad null$	null process
$\quad P \mid Q$	parallel composition
$\quad !P$	replication
$\quad$ new $a; P$	name restriction
$\quad$ let $x = D$ in P else Q	term evaluation
$\quad$ in$(M, x); P$	message input
$\quad$ out$(M, N); P$	message output
$\quad$ phase $t; P$	weak phase

We note that the ProVerif software tool allows the definition of a single main process which in turn may refer to subprocesss of the form "let $P = Q$." The tool also permits the use of comments in the form *(* comment *)*.

The choice operator allows us to model a pair of processes which have the same structure and differ only in the choice of terms and terms evaluations. We call such a pair of processes a *biprocess*. Given a biprocess P, we define two processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ as follows: $\mathsf{fst}(P)$ is obtained by replacing all occurrences of choice$[M, M']$ with M and choice$[D, D']$ with D in P. Similarly, $\mathsf{snd}(P)$ is obtained by replacing choice$[M, M']$ with M' and choice$[D, D']$ with D' in P. We define $\mathsf{fst}(D)$, $\mathsf{fst}(M)$, $\mathsf{snd}(D)$ and $\mathsf{snd}(M)$ similarly.

As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $fv(P)$, $bv(P)$ (resp. $fn(P)$ and $bn(P)$) for the sets of free and bound variables (resp. names) in P. A process is *closed* if it has no free variables (but it may contain free names). A *context* $C[\_]$ is a process with a hole. We obtain $C[P]$ as the result of filling $C[\_]$'s hole with P. An *evaluation context* C is a closed context built from $[\_]$, $C \mid P$, $P \mid C$ and new $a; C$. We sometimes refer to contexts without choice as *plain contexts*.

The major difference between the syntax of the applied pi calculus and the calculus of ProVerif, is the introduction of the choice operator. In addition there are some minor changes. For instance, communication is permitted on arbitrary terms, not just names. Function symbols are supplemented with destructors. Active substitutions are removed in favour of term evaluations. The syntax does not include the conditional "if $M = N$ then P else Q", which can be defined as "let $x = equals(M, N)$ in P else Q" where $x \notin fv(P)$ and *equals* is a destructor with the equation $equals(x, x) = x$. We omit "else Q" when the process Q is *null*. Finally the calculus of ProVerif does not rely on a sort system. We believe that processes written in the calculus of

ProVerif, can be mapped to semantically equivalent processes in the applied pi calculus and vice-versa, although proving this remains an open problem. This can easily be extended to biprocesses.

2.2 Operational semantics

The operational semantics of processes in the calculus of ProVerif, are defined by three relations, namely *term evaluation* $\Downarrow$, *structural equivalence* $\equiv$ and *reduction* $\rightarrow$. Structural equivalence and reductions are only defined on closed processes. We write $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$, and $\rightarrow^*\equiv$ for its union with $\equiv$. The operational semantics for the calculus of ProVerif differ in minor ways from the semantics of the applied pi calculus. *Structural equivalence* is the smallest equivalence relation on processes that is closed under application of evaluation contexts and some other standard rules such as associativity and commutativity of the parallel operator. *Reduction* is the smallest relation on biprocesses closed under structural equivalence and application of evaluation contexts such that

$$\begin{array}{ll}
\textsc{Red I/O} & \mathrm{out}(N,M);Q \mid \mathrm{in}(N',x);P \rightarrow Q \mid P\{^M/_x\} \\
 & \qquad \text{if } \mathsf{fst}(N)=\mathsf{fst}(N') \text{ and } \mathsf{snd}(N)=\mathsf{snd}(N') \\
\textsc{Red Fun 1} & \text{let } x=D \text{ in } P \text{ else } Q \rightarrow P\{^{\mathrm{choice}[M_1,M_2]}/_x\} \\
 & \qquad \text{if } \mathsf{fst}(D)\Downarrow M_1 \text{ and } \mathsf{snd}(D)\Downarrow M_2 \\
\textsc{Red Fun 2} & \text{let } x=D \text{ in } P \text{ else } Q \rightarrow Q \\
 & \qquad \text{if there is no } M_1 \text{ such that } \mathsf{fst}(D)\Downarrow M_1 \text{ and} \\
 & \qquad \text{there is no } M_2 \text{ such that } \mathsf{snd}(D)\Downarrow M_2 \\
\textsc{Red Repl} & !P \rightarrow P \mid !P
\end{array}$$

2.3 Extension to processes with weak phases

Many protocols can be broken into phases, and their security properties can be formulated in terms of these phases. Typically, for instance, if a protocol discloses a session key after the conclusion of a session, then the secrecy of the data exchanged during the session may be compromised but not its authenticity. To enable modelling of protocols with several phases the calculus of ProVerif is extended [3]. The syntax of processes is supplemented with a phase prefix "phase $t;P$", where t is a non-negative integer. Intuitively, t represents a global clock, and the process "phase $t;P$" is active only during phase t. However, it is possible that *not* all instructions of a particular phase are executed prior to a phase transition. Moreover, parallel processes may only communicate if they are under the same phase.

Example 1. Let $P = \text{phase } 1; \text{out}(c, a) \mid \text{phase } 2; \text{out}(c, b)$. The process P can output b without having first output a.

The semantics of processes are extended to deal with weak phases (see [3]).

2.4 Observational equivalence

The notion of observational equivalence was introduced by Abadi & Fournet [1], subsequently Blanchet, Abadi & Fournet [3] defined strong observational equivalence. This paper will use strong observational equivalence, henceforth we shall use observational equivalence to mean strong observational equivalence. We first recall the standard definition of observational equivalence. We write $P \downarrow_M$ when P emits a message on the channel M, that is, when $P \equiv C[\text{out}(M', N); R]$ for some evaluation context $C[\_]$ that does not bind $fn(M)$ and $M =_{\mathsf{E}} M'$.

Definition 1 ([3]). *Observational equivalence* $\sim$ is the largest symmetric relation $\mathcal{R}$ on closed processes such that $P \mathcal{R} Q$ implies:

1. if $P \downarrow_M$ then $Q \downarrow_M$;
2. if $P \rightarrow P'$ then there exists Q' such that $Q \rightarrow Q'$ and $P' \mathcal{R} Q'$;
3. $C[P] \mathcal{R} C[Q]$ for all evaluation contexts C.

Intuitively, a context may represent an attacker, and two processes are observationally equivalent if they cannot be distinguished by any attacker. Given a biprocess P, we say that P satisfies observational equivalence when we have that $\mathsf{fst}(P) \sim \mathsf{snd}(P)$.

A reduction $P \rightarrow Q$ for a biprocess P implies the corresponding processes have reductions $\mathsf{fst}(P) \rightarrow \mathsf{fst}(Q)$ and $\mathsf{snd}(P) \rightarrow \mathsf{snd}(Q)$. However, reductions in $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ do not necessarily correspond to any biprocess reduction. When such a corresponding reduction does exist the processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ satisfy uniformity under reduction (UUR):

Definition 2 ([3]). A biprocess P satisfies *uniformity under reduction* if:

1. $\mathsf{fst}(P) \rightarrow Q_1$ implies that $P \rightarrow Q$ for some biprocess Q with $\mathsf{fst}(Q) \equiv Q_1$, and symmetrically for $\mathsf{snd}(P) \rightarrow Q_2$;
2. for all plain evaluation contexts C, for all biprocess Q, $C[P] \rightarrow Q$ implies that Q satisfies UUR.

Blanchet *et al.* [3] have shown that if a biprocess P satisfies uniformity under reductions then P satisfies observational equivalence. The ProVerif software automatically verifies whether its input satisfies uniformity under reduction and thus enables us to prove observational equivalence in some cases.

2.5 Limitations of the calculus

There are trivial equivalences (see Example 2 described below) which the calculus of ProVerif is unable to prove since the definition of observational equivalence by uniformity under reductions is too strong. We overcome this problem with *data swapping.*

Example 2. The equivalence $\text{out}(c, a) \mid \text{out}(c, b) \sim \text{out}(c, b) \mid \text{out}(c, a)$ holds trivially since the processes are in fact structurally equivalent. But the corresponding biprocess $\text{out}(c, \text{choice}[a, b]) \mid \text{out}(c, \text{choice}[b, a])$ does not satisfy uniformity under reductions and therefore the equivalence cannot be proved by ProVerif.

Moreover, the phase semantics introduced by the calculus of ProVerif [3] are insufficient to model protocols which require synchronisation, as the phase semantics do not enforce that all instances of a phase must be completed prior to phase progression. We solve this problem with the introduction of *strong phases.*

Both of these problems are encountered when modelling cryptographic protocols from literature. As case studies we demonstrate the suitability of our approach by modelling the privacy properties of the electronic voting protocol FOO [10] and Direct Anonymous Attestation (DAA) [4].

3 Extending the calculus

To overcome the limitations stated in the previous section, we extend the calculus with strong phases and data swapping.

3.1 Extension to processes with strong phases

Similarly to weak phases the syntax of processes is supplemented with a strong phase prefix "strong phase $t; P$", where t is a non-negative integer. A strong phase represents a global synchronisation and t represents the global clock. The process strong phase $t; P$ is active only during strong phase t and a strong phase progression may only occur once all the instructions under the previous phase have been executed.

Example 3. Consider our earlier example (Example 1) with the use of strong phase. Now, the process

$$\text{strong phase } 1; \text{out}(c, a) \mid \text{strong phase } 2; \text{out}(c, b)$$

cannot output b without having previously output a.

3.2 Extension to processes with data swapping

Let us first consider the background to our approach. Referring back to Example 2 we recall the biprocess $Q = \text{out}(c, \text{choice}[a, b]) \mid \text{out}(c, \text{choice}[b, a])$ which does not satisfy UUR. We note that $\mathsf{fst}(Q) = \text{out}(c, a) \mid \text{out}(c, b)$ and $\mathsf{snd}(Q) = \text{out}(c, b) \mid \text{out}(c, a)$. Since $\text{out}(c, b) \mid \text{out}(c, a) \equiv \text{out}(c, a) \mid \text{out}(c, b)$ it seems reasonable to rewrite $\mathsf{snd}(Q)$ as $\text{out}(c, a) \mid \text{out}(c, b)$, enabling us to write Q as $\text{out}(c, \text{choice}[a, a]) \mid \text{out}(c, \text{choice}[b, b])$ which is semantically equivalent to $\text{out}(c, a) \mid \text{out}(c, b)$. Our new biprocess satisfies uniformity under reduction, and thus observational equivalence. It therefore seems possible (under certain circumstances) to *swap* values from the left to the right side of the parallel operator. Sometimes the swap is not done initially but instead immediately after a strong phase. To specify data swapping we introduce the special comment *(**swap*)* in process descriptions, which can be seen as a *proof hint.* Returning to our example, we would rewrite Q as

$$\begin{aligned} Q' &= \textit{(**swap*)}\ \text{out}(c, \text{choice}[a, b]) \mid \textit{(**swap*)}\ \text{out}(c, \text{choice}[b, a]) \\ &= \text{out}(c, \text{choice}[a, a]) \mid \text{out}(c, \text{choice}[b, b]). \end{aligned}$$

3.3 Automated reasoning with ProVerif

To allow automated reasoning we describe a translator which accepts as input processes written in our extended language. It will also include a single main process and subprocesses of the form "let $P = Q$", subject to the following restrictions.

1. The commands strong phase t; and *(**swap*)* can only appear in a single subprocess defined using the let keyword (not in the main process);
2. The subprocess defined using the let keyword that contain strong phases and data swapping must be instantiated precisely twice in the main process. Moreover, it must be of the form let $P = \alpha$, where α is a process that is sequential until its last strong phase, at which point it is an arbitrary process. Formally α is given by the grammar below:

   $$\alpha := R \big| \text{new } a; \alpha \big| \text{in}(M, x); \alpha \big| \text{out}(M, N); \alpha \big| \text{let } x = D \text{ in } \alpha \big| \text{strong phase } t; \alpha$$

   where R is an arbitrary processes without data swapping and strong phases;
3. We further require that *(**swap*)* may only occur at the start of a subprocess definition or immediately after a strong phase.

The translator outputs processes in the standard language of ProVerif, which can be automatically reasoned about by the software tool. The pseudocode of our algorithm is presented in Figure 1.

Step one of our translator makes the necessary modifications to subprocesses. It defines each strong phase as an individual subprocess. Step two handles the main process which combines the subprocesses defined in step one in such a way that preserves notion of strong phases. The other parts of the translator's input are copied to the output verbatim. We demonstrate its application with several toy examples (see Section 3.4) and two case studies (see Sections 4 & 5).

Step 1: We replace any subprocess declaration of the form

$$\text{let } P = \alpha_0;\ \text{strong phase } 1;\ \alpha_1;\ \text{strong phase } 2;\ \alpha_2;\ \ldots;\ \text{strong phase } n;\ \alpha_n.$$

with the declarations

```
let $P_0 = \alpha_0; \text{out}(pc, M_0)$.
let $P_1 = \alpha_1; \text{out}(pc, M_1)$.
    $\vdots$
let $P_{n-1} = \alpha_{n-1}; \text{out}(pc, M_{n-1})$.
let $P_n = \alpha_n$.
```

where M_i is a term consisting of a tuple containing each bound name in $\alpha_0, \alpha_1, \ldots, \alpha_i$ and the free variables in $\alpha_{i+1}, \alpha_{i+2}, \ldots, \alpha_n$.

Step 2: We replace instance declarations in the main process of the form

$$\text{let } \widetilde{x} = \widetilde{N} \text{ in } P \mid \text{let } \widetilde{x} = \widetilde{N}' \text{ in } P$$

with

```
new $pc_0$; new $pc'_0$; new $pc_1$; new $pc'_1$; ...; new $pc_{n-1}$; new $pc'_{n-1}$; (
  let $\widetilde{x} = \widetilde{N}$ in let $pc = pc_0$ in $P_0$|
  let $\widetilde{x} = \widetilde{N}'$ in let $pc = pc'_0$ in $P_0$|
  in($pc_0, z_0$); in($pc'_0, z'_0$); (* start strong phase 1 *) (
    let $M_0 = z_0$ in let $pc = pc_1$ in $P_1$|
    let $M_0 = z'_0$ in let $pc = pc'_1$ in $P_1$)|
  $\vdots$
  in($pc_{n-1}, z_{n-1}$); in($pc'_{n-1}, z'_{n-1}$); (* start strong phase n *) (
    let $M_{n-1} = z_{n-1}$ in $P_n$|
    let $M_{n-1} = z'_{n-1}$ in $P_n$)
)
```

If α_0 starts with *(**swap*)*, we further modify the above description, by replacing

let $\widetilde{x} = \widetilde{N}$ in	*with*	let $\widetilde{x} = \text{choice}[\widetilde{N}, \widetilde{N}']$ in
let $\widetilde{x} = \widetilde{N}'$ in	*with*	let $\widetilde{x} = \text{choice}[\widetilde{N}', \widetilde{N}]$ in

Similarly, if α_i starts with *(**swap*)* and $1 \leq i \leq n$, we further modify the description

let $M_i = z_i$ in	*with*	let $M_i = \text{choice}[z_i, z'_i]$ in
let $M_i = z'_i$ in	*with*	let $M_i = \text{choice}[z'_i, z_i]$ in

Fig. 1 Translator algorithm

3.4 Examples

Example 4. We begin by returning to our trivial observational equivalence:

$$\mathrm{out}(c,a) \mid \mathrm{out}(c,b) \sim \mathrm{out}(c,b) \mid \mathrm{out}(c,a).$$

As the definition of observational equivalence by UUR is too strong, the calculus, and therefore the software tool, are unable to reason about such an equivalence. Using our data swapping syntax, the biprocess encoding the previous equivalence is given below.

```
let P = (**swap*) out(c,x).
process let x = choice[a,b] in P|let x = choice[b,a] in P
```

Our translator gives us the following biprocess, which ProVerif can successfully prove.

```
let P = out(c,x).
process let x = choice[choice[a,b],choice[b,a]] in P|
        let x = choice[choice[b,a],choice[a,b]] in P
```

Example 5. We consider the observational equivalence shown below:

$$\begin{aligned} &\mathrm{out}(c,a); \text{strong phase } 1; \mathrm{out}(c,d) \mid \mathrm{out}(c,b); \text{strong phase } 1; \mathit{null} \\ \sim \quad &\mathrm{out}(c,a); \text{strong phase } 1; \mathit{null} \mid \mathrm{out}(c,b); \text{strong phase } 1; \mathrm{out}(c,d) \end{aligned}$$

The pair of processes are both able to output a and b. We then have a synchronisation and obtain the process $\mathrm{out}(c,d) \mid \mathit{null} \sim \mathit{null} \mid \mathrm{out}(c,d)$. To allow ProVerif to prove such an equivalence we provide our translator with the following input:

```
let P =out(c,x);strong phase 1;(**swap*)if y=ok then out(c,d).
process let x = a in let y = choice[ok,ko] in P|
        let x = b in let y = choice[ko,ok] in P
```

Our translator produces the biprocess described below.

```
let P1 = out(c,x);out(pc,y).
let P2 = if y = ok then out(c,c).
process new pc0;new pc1;(
  let x = a in let y = choice[ok,ko] in let pc = pc0 in P1|
  let x = b in let y = choice[ko,ok] in let pc = pc1 in P1|
  in(pc0,y0);in(pc1,y1);(
    let y = choice[y0,y1] in P2|
    let y = choice[y1,y0] in P2))
```

Example 6. As our final example we consider the following equivalence:

$$\begin{aligned} &\mathrm{out}(c,a_1); \text{strong phase } 1; \mathrm{out}(c,a_2) \mid \mathrm{out}(c,b_1); \text{strong phase } 1; \mathrm{out}(c,b_2) \\ \sim \quad &\mathrm{out}(c,a_1); \text{strong phase } 1; \mathrm{out}(c,b_2) \mid \mathrm{out}(c,b_1); \text{strong phase } 1; \mathrm{out}(c,a_2) \end{aligned}$$

This is similar to Example 5 with two outputs after the strong phase. Again, thanks to our translator, we are able to conclude on such an example.

4 E-voting protocol due to Fujioka *et al.*

In this section, we study the privacy property of the e-voting protocol due to Fujioka *et al.* [10]. In [12], it is shown that this protocol provides fairness, eligibility and privacy. However, the proof of privacy given in [12] is manual: ProVerif is unable to prove it directly.

4.1 Description

The protocol involves voters, an administrator and a collector. The administrator is responsible for verifying that only eligible voters can cast votes and the collector handles the collecting and publishing of votes. The protocol requires three strong phases.

In the first phase, the voter gets a signature on a commitment to his vote from the administrator, i.e. $m = sign(blind(commit(v,k),r),ska)$ where k is a random key, r is a blinding factor and ska is the private key of the administrator. At the end of this first phase, the voter unblinds m and obtains $y = sign(commit(v,k),ska)$, i.e. the signature of his commitment. The second phase of the protocol is the actual voting phase. The voter sends y to the collector who checks correctness of the signature and, if the test succeeds, enters (ℓ,x,y) onto a list as an ℓ-th item. The last phase of the voting protocol starts, once the collector decides that he received all votes, e.g. after a fixed deadline. In this phase the voters reveal the random key k which allows the collector to open the votes and publish them. The voter verifies that his commitment is in the list and sends ℓ, r to the collector. Hence, the collector opens the ballots.

4.2 Modelling privacy in applied pi

Privacy properties have been successfully studied using equivalences. In the context of voting protocols, the definition of privacy is rather subtle. We recall the definition of privacy for electronic voting protocols given in [12]. A voting protocol guarantees ballot secrecy (privacy) whenever a process where Alice votes for candidate v_1 and Bob votes for candidate v_2 is observationally equivalent to a process where their votes are swapped, i.e. Alice votes v_2 and Bob votes v_1. We denote their secret keys $skva$ and $skvb$ respectively. In [12],

```
let V =
 new k;new r;
 let x = commit(v,k) in
 out(c,(pk(skv),sign(blind(x,r),skv)));
 in(c,m2);
 let y = unblind(m2,r) in
 if checksign(y,pka) = x then
 strong phase 1; (**swap*)
 out(c,y);
 strong phase 2;
 in(c,(l,yprime));
 if yprime = y then out(c,(l,k)).

process
 new ska;new skva;new skvb;
 let pka = pk(ska) in
 out(c,(ska,pka,pk(skva),pk(skvb)));(
  (let (skv,v) = (skva,choice[v1,v2]) in V)|
  (let (skv,v) = (skvb,choice[v2,v1]) in V))
```

Process 1 FOO model (extended syntax)

they rely on hand proof techniques to show privacy on FOO. Our modelling of FOO in the applied pi is similar to the one given in [8]. The underlying equational theory is the same as in [12].

The main process given in Process 1 models the environment and specifies how the other processes are combined. To establish privacy, we do not require that the authorities are honest, so we do not need to model them and we only consider two voter processes in parallel. First, fresh private keys for the voters and the administrator are generated. The corresponding public keys are then made available to the attacker. We also output the secret key of the administrator. This allows the environment to simulate the administrator (even a corrupted one) and hence we show that the privacy property holds even in the presence of a corrupt administrator.

The process V given in Process 1 models the role of a voter. The specification follows directly from our informal description. Note that we use the strong phase command to enforce the synchronisation of the voter processes. As mentioned initially in [12], the separation of the protocol into strong phases is crucial for privacy to hold. We also provide a data swapping hint to allow our translator to produce an output suitable for automatic verification using ProVerif.

4.3 Analysis

We use our translator to remove all instances of strong phases and handle data swapping. Our translator produces Process 2, which is suitable for automatic

```
let V1 =
 new k;new r;
 let x = commit(v,k) in
 out(c,(pk(skv),sign(blind(x,r),skv)));
 in(c,m2);
 let y = unblind(m2,r) in
 if checksign(y,pka) = x then out(pc,(y,k)).

let V2 =
 out(c,y); out(pc,(y,k)).

let V3 =
 in(c,(l,yprime)); if yprime = y then out(c,(l,k)).

process
 new ska;new skva;new skvb;
 let pka = pk(ska) in
 out(c,(ska,pka,pk(skva),pk(skvb)));
 new pc1;new pc2;new pc3;new pc4;(
  (let (skv,v)=(skva,choice[v1,v2]) in let pc=pc1 in V1)|
  (let (skv,v)=(skvb,choice[v2,v1]) in let pc=pc2 in V1)|
  (in(pc1,(y1,k1));in(pc2,(y2,k2));(*strong phase 1*)(*swap*)(
   (let (y,k)=choice[(y1,k1),(y2,k2)] in let pc=pc3 in V2)|
   (let (y,k)=choice[(y2,k2),(y1,k1)] in let pc=pc4 in V2)))|
  (in(pc3,(y3,k3));in(pc4,(y4,k4));(*strong phase 2*)(
    (let (y,k)=(y3,k3) in V3)|
    (let (y,k)=(y4,k4) in V3))))
```

Process 2 Translated FOO model (ProVerif syntax)

verification using ProVerif. Hence, using our approach, we provide the first automatic and complete proof that this protocol satisfies privacy.

5 Direct Anonymous Attestation (DAA)

The Direct Anonymous Attestation (DAA) scheme provides a means for remotely authenticating a trusted platform whilst preserving the user's privacy [4]. In [15], two of the authors have shown that corrupt administrators are able to violate the privacy of the host. Using our extended calculus we are now able to provide a formal and automatic proof that the rectified protocol proposed in [15] satisfies its privacy requirements. We start with a short description of the protocol (for a more complete description, see [4, 15]).

5.1 Description

The protocol can be seen as a group signature scheme without the ability to revoke anonymity and an additional mechanism to detect rogue members. In broad terms the *host* contacts an *issuer* and requests membership to a group. If the issuer wishes to accept the request, it grants the host/TPM an *attestation identity credential*. The host is now able to anonymously authenticate itself as a group member to a *verifier* with respect its credential.

The protocol is initiated when a host wishes to obtain a credential. This is known as the join protocol. The TPM creates a secret f value and a blinding factor v', where $f = hash(hash(\texttt{DAASeed}\|hash(PK_I'))\|\texttt{cnt}\|0)$. The value `DAASeed` is a secret known only to the TPM, `cnt` is a counter used by the TPM to keep track of how many times the Join protocol has been run and PK_I' is the long term public key of the issuer. The inclusion of PK_I' prevents cross issuer linkability [15]. The TPM then constructs the blind message $U := blind(f, v')$ and $N_I := \zeta_I^f$, where $\zeta_I := hash(0\|bsn_I)$ and bsn_I is the basename of the issuer (see [15] for further discussion on DAA basenames). The U and N_I values are submitted to the issuer I. The issuer creates a random nonce value n_e, encrypts it with the public key PK_{EK} of the host's TPM and returns the encrypted value. The TPM decrypts the message, revealing n_e, and returns $hash(U\|n_e)$. The issuer confirms that the hash is correctly formed. The issuer generates a nonce n_i and sends it to the host. The host/TPM constructs a signature proof of knowledge that the messages U and N_I are correctly formed. The issuer verifies the proof and generates a blind signature on the message U. It returns the signature along with a proof that a covert channel has not been used. The host verifies the signature and proof and the TPM unblinds the signature revealing a secret credential v (the signed f).

Once the host has obtained an anonymous attestation credential from the issuer it is able to produce a signature proof of knowledge of attestation on a message m. This is known as the sign/verify protocol. The verifier sends nonce n_v to the host. The host/TPM produce a signature proof of knowledge of attestation on the message $(n_t\|n_v\|b\|m)$, where n_t is a nonce defined by the TPM and b is a parameter. In addition the host computes $N_V := \zeta^f$, where $\zeta := hash(1\|bsn_V)$ and bsn_V is the basename of the verifier. Intuitively if a verifier is presented with such a proof it is convinced that it is communicating with a trusted platform and the message is genuine.

5.2 Modelling privacy in applied pi

The DAA protocol satisfies privacy whenever a process where Alice interacts with the verifier is observationally equivalent to when Bob interacts with the verifier. For privacy we require that both Alice and Bob have completed the join protocol.

Signature and equational theory. The DAA protocol makes extensive use of signature proofs of knowledge (SPK) to prove knowledge of and relations among discrete logarithms. We will discuss our formalism with an example. The signature proof of knowledge $\mathrm{SPK}\{(\alpha, \beta) : x = g^{\alpha} \wedge y = h^{\beta}\}(m)$ denotes a signature proof of knowledge on the message m that x, y were constructed correctly. This leads us to define function `spk/3` to construct an SPK. The first argument contains a tuple of secret values known to the prover α, β. The second argument consists of a tuple of the values on which the prover is claiming to have constructed correctly x, y, such that $x = g^{\alpha}$ and $y = h^{\beta}$. Finally the third argument is the message m on which the prover produces a signature on. Verifying the correctness of a SPK is specific to its construction, thus we must require a function `checkspk` for each SPK that the protocol uses. To verify the SPK produced in the aforementioned example the verifier must be in possession of the SPK itself and x, y, g, h, m. We define the equation: $checkspk(spk((\alpha, \beta), (g^{\alpha}, h^{\beta}), m), g^{\alpha}, h^{\beta}, g, h, m) = ok$. A verifier can now check a SPK using an `if` statement.

Modelling the DAA protocol. As in FOO, the main process (see [9]) models the environment and specifies how the other processes are combined. First, fresh secret keys for the TPMs, the issuer and the verifier are generated using the restriction operator. We also generate two `DAASeed` values. The public keys are then sent on a public channel, i.e. they are made available to the intruder. We also output the secret key of the verifier and issuer since the privacy property should be preserved even if they are corrupt. Next we input the basenames bsn_I, bsn_V of the issuer and verifier. Then we instantiate two instances of the DAA protocol with the necessary parameters.

Our encoding of the DAA protocol (see [9]) follows directly from our informal description. Note that we use the strong phase and data swapping commands introduced by our extension to the calculus to ensure synchronisation. The two instances of the DAA processes must first execute all instructions of DAAJoin before moving onto DAASign. The separation of the protocol into strong phases is crucial for privacy to hold.

5.3 Analysis

We use our translator to remove all instances of strong phases from our encoding and produce code suitable for input to ProVerif. Our translator produces a process (see [9]) which permits the automatic verification of the privacy property using ProVerif. We are also able to detect the vulnerability in the original DAA protocol and prove the optimisation presented in [15].

6 Conclusion

In this paper we have extended the class of equivalences which ProVerif is able to automatically verify. More specifically we are able to reason about processes which require data swapping and/or strong phases. Using the approach developed we are able to automatically verify the privacy properties of the electronic voting protocol FOO and the Direct Anonymous Attestation scheme. In future work, we would like to generalise our translation algorithm and provide a formal proof of the correctness of our translator. Moreover we plan to automate the swapping procedure.

References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL'01: Proc. 28th ACM Symposium on Principles of Programming Languages, pp. 104–115. ACM Press, New York, USA (2001)
2. 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, Proceedings of SSP'08 (2008). To appear
3. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming **75**(1), 3–51 (2008)
4. Brickell, E., Camenisch, J., Chen, L.: Direct Anonymous Attestation. In: CCS '04: 11th ACM conference on Computer and communications security, pp. 132–145. ACM Press, New York, USA (2004)
5. Chadha, R., Kremer, S., Scedrov, A.: Formal Analysis of Multi-Party Fair Exchange Protocols. In: R. Focardi (ed.) 17th IEEE Computer Security Foundations Workshop, pp. 266–279. IEEE, Asilomar, USA (2004)
6. Clark, J., Jacob, J.: A Survey of Authentication Protocol Literature (1997). URL http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps
7. Delaune, S., Kremer, S., Ryan, M.: Coercion-Resistance and Receipt-Freeness in Electronic Voting. In: CSFW '06: Proc. 19th IEEE workshop on Computer Security Foundations, pp. 28–42. IEEE (2006)
8. Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Research report, Laboratoire Spécification et Vérification, ENS Cachan, France (2008)
9. Delaune, S., Ryan, M., Smyth, B.: Automatic verification of privacy properties in the applied pi calculus (extended version) (2008). URL http://www.cs.bham.ac.uk/~bas/
10. Fujioka, A., Okamoto, T., Ohta, K.: A Practical Secret Voting Scheme for Large Scale Elections. In: ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques, pp. 244–251. Springer, London (1993)
11. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: Eurocrypt, *LNCS*, vol. 1807, pp. 539–556 (2000)
12. Kremer, S., Ryan, M.D.: Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In: ESOP'05: Proc. of the European Symposium on Programming, *LNCS*, vol. 3444, pp. 186–200 (2005)
13. Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. Information Processing Letters **56**(3), 131–133 (1995)
14. Mukhamedov, A., Ryan, M.D.: Fair Multi-party Contract Signing using Private Contract Signatures. Information & Computation (2007). Preprint
15. Smyth, B., Ryan, M., Chen, L.: Direct Anonymous Attestation (DAA): Ensuring privacy with corrupt administrators. In: ESAS'07: 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks, *LNCS*, vol. 4572, pp. 218–231 (2007)