Automatically Checking Commitment Protocols in ProVerif without False Attacks

Tom Chothia[1], Ben Smyth[2], and Chris Staite[1]

[1] School of Computer Science, University of Birmingham, UK
[2] Mathematical and Algorithmic Sciences Lab, France Research Center, Huawei Technologies Co. Ltd., France

Abstract. ProVerif over-approximates the attacker's power to enable verification of processes under replication. Unfortunately, this results in ProVerif finding false attacks. This problem is particularly common in protocols whereby a participant commits to a particular value and later reveals their value. We introduce a method to reduce false attacks when analysing secrecy. First, we show how inserting phases into non-replicated processes enables a more accurate translation to Horn clauses which avoids some false attacks. Secondly, we generalise our methodology to processes under replication. Finally, we demonstrate the applicability of our technique by analysing BlueTooth Simple Pairing. Moreover, we propose a simplification of this protocol that achieves the same security goal.

1 Introduction

State space exploration has emerged as a leading verification technique [25] and, in this context, Abadi & Fournet [2] propose the applied pi calculus – an extension of the pi calculus – to reason with cryptographic protocols. Unfortunately, proving security in this context is undecidable [21], due to several sources of unboundedness, including, messages of arbitrary length and the possibility of an unbounded number of sessions. Accordingly, state-of-the-art automated reasoning techniques focus on sound, but incomplete, methodologies, which may report false attacks and do not always terminate.

Blanchet [8,9,10] translates applied pi calculus processes to Horn clauses and uses resolution of Horn clauses to reason with secrecy and authentication properties, these results have been implemented in ProVerif [15]. ProVerif has been successfully used to automatically analyse cryptographic protocols from a variety of applications domains, including, key exchange [1,10,29], electronic voting [20,5,4] and trusted computing [18,19,28], for example. However, Blanchet's translation to Horn clauses over-approximates the attacker's power and, therefore, ProVerif may report false attacks; as highlighted by Blanchet [12, §2.2]:

⋆ A long version of this paper and ProVerif source code supporting this paper are available from `http://www.cs.bham.ac.uk/~tpc/projects/falseattacks`.

R. Focardi and A. Myers (Eds.): POST 2015, LNCS 9036, pp. 137–155, 2015.
DOI: 10.1007/978-3-662-46666-7_8

> *"false attacks occur typically for protocols that first need to keep data secret, then publish them later in the protocol. In that situation, the Horn clause model considers that the attacker can re-inject the secret in the early part of the run, which is not possible in reality."*

This behaviour is typical of protocols in which participants commit to a value and later reveal it, such as the BlueTooth Simple Pairing protocol [22,23], which we analyse in Section 6. In this paper, we introduce techniques for analysing secrecy that avoids some false attacks.

1.1 Our Contribution in Context of Existing Work

Let us consider the following process, proposed by Allamigeon & Blanchet [3, §3.2]:

$$\nu\, n.c(x).\overline{c}\langle n\rangle.\text{if } x = n \text{ then } \overline{c}\langle s\rangle \tag{P1}$$

Process P1 generates a fresh bound name n, binds a message input to variable x, outputs the name n, tests if the message bound to x is equal to n, and outputs the free name s, if the test succeeds. It follows intuitively that an attacker that does not know s in advance, cannot derive s from Process P1, that is, we have secrecy$(\{c\}, s)$: P1. However, the Horn clauses generated by Blanchet's translation of P1 include:

$$\texttt{attacker}(x) \Rightarrow \texttt{attacker}(n)$$
$$\texttt{attacker}(n) \Rightarrow \texttt{attacker}(s)$$

Hence, ProVerif cannot prove secrecy$(\{c\}, s)$: P1, because the Horn clauses permit the following false attack: knowledge of an arbitrary term M implies knowledge of n and knowledge of n implies knowledge of s. This is due to an over-approximation: the Horn clauses model the process $\nu\, n.!c(x).\overline{c}\langle n\rangle.\text{if } x = n \text{ then } \overline{c}\langle s\rangle$, rather than P1. It follows that the Horn clauses do not enforce that a message input must be received *before* the bound name n is output. Intuitively, such false attacks can be avoided by ensuring that the translation preserves temporal order of message inputs and outputs. We shall achieve this objective using Blanchet, Abadi & Fournet's notion of *phases* [13, §8]. Phases, denoted $t\!:\!P$, ensure which parts of concurrent processes are active at a particular time.

Inserting phases into a process can stop false attack, e.g., we can add a phase into process P1:

$$0\!:\!\nu\, n.c(x).1\!:\!\overline{c}\langle n\rangle.\text{if } x = n \text{ then } \overline{c}\langle s\rangle \tag{P2}$$

The semantics of phases ensure that P2 is a sound approximation of P1. However, the Horn clauses generated by Blanchet's translation of P2 to Horn clauses are more precise:

$$\texttt{attacker}(x) \Rightarrow \texttt{attacker}'(x)$$
$$\texttt{attacker}(x) \Rightarrow \texttt{attacker}'(n)$$
$$\texttt{attacker}(n) \Rightarrow \texttt{attacker}'(s)$$

Indeed, ProVerif can use these Horn clauses to prove secrecy$(\{c\}, s)$: P2 and, since P2 is a sound approximation of P1, we have secrecy$(\{c\}, s)$: P1.

We define a compiler (Section 3) that inserts phases into a restricted class of processes such that Blanchet's translation from compiled processes to Horn clauses enforces order. We prove the soundness of our methodology:

secrecy holds in the original process
iff secrecy holds in the compiled process

This technique is limited to proving secrecy of names which are not under the scope of a replication and we overcome this limitation using an alternative notion of secrecy.

The class of secrecy properties which can be considered using Blanchet's definition [10, Section 3.1] is limited, for example, we cannot consider secrecy of an arbitrary session secret s in the process $!\nu\, s.P$, that is, we cannot consider if an instance of $\nu\, s.P$ leaks s. This problem can be overcome by abstraction, in particular, Blanchet [10, §2.3] proposes the following solution. Extend the set of function symbols with the binary constructor e and the binary destructor d, let the set of rewrite rules $\mathrm{def}(d) = \{d(x, e(x, y)) \to y\}$, and modify $!\nu\, s.P$ such that $e(s, m)$ is published at the end of every *successful* session of $\nu\, s.P$, where m is a free name not known by the attacker. It follows that the modified process preserves secrecy of m iff $!\nu\, s.P$ preserves secrecy of s, hence, we have a methodology to consider the secrecy of bound names.

We consider (Section 4) a definition for secrecy of bound names which does not require abstraction and introduce a new proof technique: given a process P, name s, and fresh name s', we have

secrecy of the bound name s in $!\nu\, s.P$
iff secrecy of the bound name s' in $\nu\, s'.(P\{^{s'}/_{s}\}) \mid !\nu\, s.P)$
iff secrecy of the free name s' in $P\{^{s'}/_{s}\} \mid !\nu\, s.P$

In the context of these results, we describe how our compiler can be applied to avoid false attacks, in particular, we can prove security results for secrets under the scope of a replication.

We demonstrate the applicability of our technique by analysing three protocols (Sections 5 & 6): a toy extension of the Needham-Schroeder protocol in which one of the participants reveals their nonce at the end of a successful run, the Bluetooth Simple Pairing [22,23], and a simplification of the Bluetooth Simple Pairing that we propose. ProVerif finds false attacks against each of these protocols, whereas our techniques allow us to prove security.

Pairing protocols typically use a low-entropy, human-verifiable string, derived from a high-entropy shared secret, to authenticate protocol participants and protect against impersonation attacks. Given that the string is low-entropy, strings derived from distinct secrets may collide and an attacker that can predict collisions can launch impersonation attacks. Accordingly, pairing protocols must ensure that deriving collisions is computationally expensive. The Bluetooth Simple

Pairing protocol defends against such attacks by making both parties commit to particular values before the low-entropy string can be calculated; this deprives an attacker of the opportunity to carry out a brute force attack to find a collision.

We develop theory to enable protocols that are vulnerable to collision attacks to be accurately modelled in the applied pi calculus. Accordingly, our analysis of Bluetooth Simple Pairing is more precise than an earlier analysis by Chang & Shmatikov [17], which ignores collision attacks, and so would incorrectly find the protocol to be secure even if the steps that stop impersonation attacks were removed. We also present a simplified version of the Bluetooth Simple Pairing protocol, which achieves the same secrecy goals with fewer steps, and we use our analysis method to show that it is secure.

The key contributions of this paper are:

- A framework for avoiding some false attacks when analysing secrecy.
- A definition for secrecy of bound names.
- A method to captured collision attacks.
- A demonstration of how BlueTooth Simple Pairing defends against collision attacks.
- A simplified pairing protocol.

Hence, our paper advances automated analysis techniques.

2 Background: Applied pi Calculus

We adopt Blanchet's dialect [10] of the applied pi calculus [2,27], which is suited to automated reasoning using Blanchet's ProVerif [15]. The dialect uses the notion of configurations proposed by Baudet [7] to avoid structural equivalence, which simplifies security definitions and subsequent proofs.

The calculus assumes an infinite set of *names*, an infinite set of *variables*, and a finite set of *function symbols* (*constructors* and *destructors*), each with an associated arity. We write f for a constructor, g for a destructor; constructors are used to build terms whereas destructors are used to manipulate terms in processes. *Terms* range over names, variables, and applications of constructors to terms. *Substitutions* $\{^{M}/_{x}\}$ replace the variable x with the term M. Arbitrarily large substitutions can be written as $\{^{M_1}/_{x_1}, \ldots, ^{M_n}/_{x_n}\}$ and the letters σ and τ range over substitutions. We write $M\sigma$ for the result of applying σ to the variables of M.

The signature Σ is equipped with a finite set of equations of the form $M = N$ and we derive an equational theory from this set by reflexive, symmetric and transitive closure, closure under the application of constructors, closure under substitution of terms for variables, and closure under bijective renaming of names. We write $\Sigma \vdash M = N$ for an equality modulo the equational theory and $\Sigma \vdash M \neq N$ for an inequality modulo the equational theory. (We write $M = N$ and $M \neq N$ for syntactic equality and inequality, respectively.)

The semantics of a destructor g of arity l is given by a finite set $\mathrm{def}(g)$ of rewrite rules $g(M'_1, \ldots, M'_l) \to M'$, where $M'_1, \ldots, M'_l, M'$ are terms that contain only constructors and variables; the variables of M' must be bound in $M'_1, \ldots, M'_l$ and variables are subject to renaming. The value $g(M_1, \ldots, M_l)$ is defined if and only if there exists a substitution σ and a rewrite rule $g(M'_1, \ldots, M'_l) \to M'$ in $\mathrm{def}(g)$ such that $M_i = M'_i\sigma$ for all $i \in \{1, \ldots, l\}$, and in this case $g(M_1, \ldots, M_l)$ is defined as $M'\sigma$.

The grammar for terms and *processes* is presented in Figure 1, where t is a non-negative integer representing a global clock. The process *let* $x = g(M_1, ..., M_l)$ *in* P *else* Q tries to evaluate $g(M_1, ..., M_l)$; if this succeeds (that is, if $g(M_1, ..., M_l)$ is defined), then x is bound to the result and P is executed, otherwise, Q is executed. The statement let $x = g(M_1, \ldots, M_l)$ in P else Q may be abbreviated as let $x = g(M_1, \ldots, M_l)$ in P, when Q is 0. The syntax does not include the conditional if $M = N$ then P else Q, but this can be defined as let $x = \mathsf{eq}(M, N)$ in P else Q, where x is a fresh variable, eq is a binary destructor, and $\mathrm{def}(\mathsf{eq}) = \{\mathsf{eq}(x, x) \to x\}$; we always include eq in our set of function symbols. For convenience, we may write if $M = N$ then P else Q for let $x = \mathsf{eq}(M, N)$ in P else Q and if $M = N$ then P for let $x = \mathsf{eq}(M, N)$ in P. In Figure 1, we extend Blanchet's syntax [10] with Blanchet, Abadi & Fournet's notion of *phases* [13, §8], denoted $t\!:\!P$, which ensures a process $t\!:\!P$ is only active during time t.

Fig. 1. Syntax for terms and processes

$M, N ::=$	terms
x, y, z	variables
a, b, c, k, s	names
$f(M_1, \ldots, M_n)$	constructor application
$D ::= g(M_1, \ldots, M_n)$	destructor application
$P, Q ::=$	processes
0	nil
$\overline{M}\langle N\rangle.P$	output
$M(x).P$	input
$P \mid Q$	parallel composition
$!P$	replication
$\nu\, a.P$	restriction
let $x = D$ in P else Q	term evaluation
$t\!:\!P$	phase

The sets of free and bound names, respectively variables, in process P are denoted by $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$, respectively $\mathrm{fv}(P)$ and $\mathrm{bv}(P)$. We also write $\mathrm{fn}(M)$ and $\mathrm{fv}(M)$ for the sets of names and variables in term M. A process P is closed if it has no free variables. A context C is a process with a hole and we obtain $C[P]$ as the result of filling C's hole with P. An evaluation context is a context whose hole is not in the scope of a replication, an input, an output, or a term evaluation.

The operational semantics (Figures 2) for the applied pi-calculus are defined by reduction ($\rightarrow$) on *configurations*. A configuration $\mathcal{C}$ is a pair $E, \mathcal{P}$ such that E is a finite set of names, and $\mathcal{P}$ is a finite multiset of pairs of closed process. The set E contains all the free names in $\mathcal{P}$, and is extended to include any names introduced during reduction, namely, those names introduced by $(E, \mathcal{P} \cup \{t : \nu\, a; P\}) \rightarrow (E \cup \{a'\}, \mathcal{P} \cup \{t : P\{a'/a\}\})$. A sequence of reductions, denoted $\mathcal{C}_1 \rightarrow \mathcal{C}_2 \rightarrow \cdots \rightarrow \mathcal{C}_n$, is called a *trace*. We occassionally write $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$.

Fig. 2. Operational semantics

$$E, \mathcal{P} \cup \{t{:}0\} \rightarrow E, \mathcal{P} \qquad \text{(Red Nil)}$$

$$E, \mathcal{P} \cup \{t{:}!P\} \rightarrow E, \mathcal{P} \cup \{t{:}!P, t{:}P\} \qquad \text{(Red Repl)}$$

$$E, \mathcal{P} \cup \{t{:}(P \mid Q)\}) \rightarrow E, \mathcal{P} \cup \{t{:}P, t{:}Q\} \qquad \text{(Red Par)}$$

$$E, \mathcal{P} \cup \{t{:}\nu\, a.P\} \rightarrow E \cup \{a'\}, \mathcal{P} \cup \{t{:}P\{a'/a\}\} \qquad \text{(Red Res)}$$
for some name $a' \notin E$

$$E, \mathcal{P} \cup \{t{:}\overline{N}\langle M\rangle.P, t{:}N(x).Q\} \rightarrow E, \mathcal{P} \cup \{t{:}P, t{:}Q\{M/x\}\} \qquad \text{(Red I/O)}$$

$$E, \mathcal{P} \cup \{t{:}\text{let } x = D \text{ in } L\} \rightarrow E, \mathcal{P} \cup \{t{:}P\{M/x\}\} \qquad \text{(Red Destr 1)}$$
if there exists M such that $D \rightarrow M$

$$E, \mathcal{P} \cup \{t{:}\text{let } x = D \text{ in } P \text{ else } Q\}) \rightarrow E, \mathcal{P} \cup \{t{:}Q\} \qquad \text{(Red Destr 2)}$$
if there is no M such that $D \rightarrow M'$

$$E, \mathcal{P} \cup \{t{:}t'{:}P\} \rightarrow E, \mathcal{P} \cup \{t'{:}P\} \qquad \text{(Red Order)}$$
if $t < t'$

Given a process P in the applied pi-calculus without phases and a set of names *Init*, the configuration $\mathit{Init}, \{0 : P\}$ will reduce using our semantics in exactly the same way as the configuration $\mathit{Init}, \{P\}$ using Blanchet's semantics [10]. We note that Blanchet, Abadi & Fournet [13, §8], who introduce phases, assume any process without phases is assumed to run in phase zero. In this paper, we wish to distinguish between processes in phase zero and processes without phases, so we always make the phases explicit, except in case studies where we adopt Blanchet, Abadi & Fournet's convention for brevity.

3 Secrecy of Free Names

We recall (Definition 1) Blanchet's formalisation [10] of knowledge derivable from a trace – that is, a reduction on a configuration – as any names which are output.

Definition 1. *Let* $\mathcal{T} = E_0, \mathcal{P}_0 \rightarrow^* E', \mathcal{P}'$ *be a trace,* n *be a name, and Init be a finite set of names. We write* $\text{attacker}(Init, n) : \mathcal{T}$ *if* $\mathcal{T}$ *contains a reduction* $E, \mathcal{P} \cup \{t{:}\overline{c}\langle n\rangle.P, t{:}c(x).Q\} \rightarrow E, \mathcal{P} \cup \{t{:}P, t{:}Q\{^n/_x\}\}$ *for some* $E, \mathcal{P}, P, Q, x, t$ *and* $c \in Init$.

It follows naturally that a configuration preserves the secrecy of a name if no (adversarial) process added to the configuration can generate a trace which permits the name to be derived. We recall (Definition 2) Blanchet's definition [10] for secrecy of free names.

Definition 2. *Let* $\mathcal{C} = E, \mathcal{P}$ *be a configuration,* n *be a name and Init be a finite set of names, where* $\text{fn}(\mathcal{P}) \subseteq E$. *We write* $\text{secrecy}(Init, n) : \mathcal{C}$ *if for all processes* Q *such that* $\text{fn}(Q) \subseteq Init$ *there is no trace* $\mathcal{T} = E \cup Init \cup \text{fn}(n), \mathcal{P} \cup \{Q\} \rightarrow^* E', \mathcal{P}'$ *such that* $\text{attacker}(Init, n) : \mathcal{T}$ *for some* E' *and* $\mathcal{P}'$.

Let P *be a closed process,* n *be a name, and Init be a finite set of names. We write* $\text{secrecy}(Init, n) : P$ *if* $\text{secrecy}(Init, n) : \text{fn}(P), \{P\}$.

Definition 2 facilitates the analysis of secrecy when the secret is a free name and Section 4 proposes a definition which supposes the secret is bound.

As discussed in Section 1.1, we encounter false attacks when analysing secrecy of free names and we overcome this problem in the remainder of this section.

3.1 Our Compiler: Phases Improve Horn Clause Generation

As demonstrated in Section 1.1, false attacks can be avoided by inserting phases into processes. Formally, we insert phases using function δ:

Definition 3. *Given a set of names Init and a process, we define* δ *as follows:*

$$\begin{aligned}\delta(Init, P) = \{P\} &\cup \{C[1{:}M(x).P'] : P = C[M(x).P']\} \\ &\cup \{C[1{:}\overline{M}\langle N\rangle.P'] : P = C[\overline{M}\langle N\rangle.P'] \wedge \text{fn}(M) \cap Init = \emptyset\}\end{aligned}$$

Function δ outputs a set of processes representing all ways of inserting a phase into P such that the phase appears immediately before an input or an output on a private channel[1].

The insertion of one phase does not generally result in a sound abstraction. For instance, process $\text{P2} \mid 0{:}!\text{P1}$ is not a sound approximation of $0{:}\text{P1} \mid 0{:}!\text{P1}$, because the phase in P2 prevents the instance of n generated by P2 being input by $0{:}\text{P1}$. This problem can be overcome by ensuring that all inputs and outputs are available in either phase 0 or phase 1, which can be achieved using our compiler (Definition 4). For simplicity, we restrict our compiler to multisets of processes $\mathcal{P} = \{0{:}L, 0{:}!L_1, \ldots, 0{:}!L_m\}$, where $L, L_1, \ldots, L_m$ are *linear processes* (Figure 3) and ProVerif discovers false attacks arising from process L. This is sufficient to avoid the false attacks discovered in the examples in this paper.

[1] Outputs on public channels can be received by the environment in phase 0 and replayed in phase 1, therefore function δ does not insert phases immediately before outputs on public channels.

Fig. 3. Syntax for linear processes

$L ::=$	linear processes
0	nil
$\overline{M}\langle N\rangle.L$	output
$M(x).L$	input
$\nu\, a.L$	restriction
let $x = g(M_1, \ldots, M_n)$ in L else 0	destructor application

Definition 4. *Given a set of names Init and a multiset of process* $\mathcal{P} = \{0 : L, 0 : !L_1, \ldots, 0 : !L_m\}$*, we define* Δ *as follows:*

$$\Delta(\mathit{Init}, \mathcal{P}) = \left\{ \{P\} \cup \mathcal{Q} \;\middle|\; P \in \delta(\mathit{Init}, 0 : L) \wedge \mathcal{Q} = \bigcup_{1 \le i \le m} \delta(\mathit{Init}, 0 : !L_i) \right\}$$

The compiler tries to avoid false attacks by inserting a phase into process L (in a different place for each member of the set produced by Δ). To ensure that all of the original reductions are still possible, our compiler also generates a copy of every other process with a phase in every necessary position.

3.2 Automated Reasoning without False Attacks

Our compiler is designed such that Blanchet's translation from compiled processes to Horn clauses ensures that the clauses abide by an ordering, thereby avoiding the false attacks described in Section 1.1, whilst preserving secrecy:

Theorem 1. *Given a name* s*, sets of names* E *and Init and a multiset of processes* $\mathcal{P} = \{0 : L, 0 : !L_1, \ldots, 0 : !L_m\}$*, such that* $s \notin \bigcup_{1 \le i \le n} (\mathrm{fn}(L_i) \cup \mathrm{bn}(L_i))$ *and* $\mathrm{fn}(\mathcal{P}) \subseteq E$*, we have for all* $\mathcal{Q} \in \Delta(\mathit{Init}, \mathcal{P})$ *that:*

$$\mathrm{secrecy}(\mathit{Init}, s) : E, \mathcal{P} \Leftrightarrow \mathrm{secrecy}(\mathit{Init}, s) : E, \mathcal{Q}$$

The proof of Theorem 1 appears in the long version of this paper. We demonstrate an application of Theorem 1 with reference to processes P1 and P2:

Example 1. Let $C = 0 : \nu\, n.c(x).\_$ and witness that $0 : \mathrm{P1} = C[\overline{c}\langle n\rangle.\mathrm{if}\ x = n\ \mathrm{then}\ \overline{c}\langle s\rangle]$ and $\mathrm{P2} = C[1 : \overline{c}\langle n\rangle.\mathrm{if}\ x = n\ \mathrm{then}\ \overline{c}\langle s\rangle]$, i.e., $\{\mathrm{P2}\} \in \Delta(\{c\}, \{\mathrm{P1}\})$. We have $\mathrm{secrecy}(\{c\}, s) : \{c, s\}, \{0 : \mathrm{P1}\} \Leftrightarrow \mathrm{secrecy}(\{c\}, s) : \{c, s\}, \{\mathrm{P2}\}$ by Theorem 1, hence, $\mathrm{secrecy}(\{c\}, s) : (0 : \mathrm{P1}) \Leftrightarrow \mathrm{secrecy}(\{c\}, s) : \mathrm{P2}$ by Definition 2. Moreover, since ProVerif can prove $\mathrm{secrecy}(\{c\}, s) : \mathrm{P2}$, we have the desired result, namely $\mathrm{secrecy}(\{c\}, s) : (0 : \mathrm{P1})$.

4 Secrecy of Bound Names

We adapt the notion of knowledge derivable from a trace (Definition 1) to consider names which are bound by the trace's initial configuration.

Definition 5. *Let $\mathcal{T} = E_0, \mathcal{P}_0 \rightarrow^* E_1, \mathcal{P}_1$ be a trace, n be a name, and Init be a finite set of names. We write $\widehat{\text{attacker}}(Init, n) : \mathcal{T}$ if $\mathcal{T}$ contains the following reductions*

$$\begin{aligned} E, \mathcal{P} \cup \{t{:}\,\nu\, n.P\} &\rightarrow E \cup \{n'\}, \mathcal{P} \cup \{t{:}\,P\{^{n'}/_{n}\}\} \\ &\rightarrow^* E', \mathcal{P}' \cup \{s{:}\,\overline{c}\langle n'\rangle.P', s{:}\,c(x).Q'\} \\ &\rightarrow E', \mathcal{P}' \cup \{s{:}\,P', s{:}\,Q'\{^{n'}/_{x}\}\} \end{aligned}$$

where $c \in Init$ and some $E, E', \mathcal{P}, \mathcal{P}', P, P', Q', n', x, s$ and t.

Intuitively, a trace $\mathcal{T}$ satisfies $\widehat{\text{attacker}}(Init, n) : \mathcal{T}$ if n is bound by the trace's initial configuration, n is renamed to n' and, subsequently, n' is outputted.

It follows naturally from Definition 5 that a configuration preserves secrecy of a bound name if no (adversarial) process added to the configuration can generate a trace which permits the name to be derived.

Definition 6. *Let $\mathcal{C} = E, \mathcal{P}$ be a configuration, n be a name and Init be a set of names, where $\text{fn}(\mathcal{P}) \subseteq E$. We write $\widehat{\text{secrecy}}(Init, n) : \mathcal{C}$ if for all processes Q such that $\text{fn}(Q) \subseteq Init$ and $n \notin \text{bn}(Q)$, there is no trace $\mathcal{T} = E \cup Init, \mathcal{P} \cup \{Q\} \rightarrow^* E', \mathcal{P}'$ such that $\widehat{\text{attacker}}(Init, n) : \mathcal{T}$ for some E' and $\mathcal{P}'$.*

Let P be a closed process, n be a name and Init be a finite set of names, where $n \in \text{bn}(P)$. We write $\widehat{\text{secrecy}}(Init, n) : P$ if $\widehat{\text{secrecy}}(Init, n) : \text{fn}(P), \{P\}$.

We remark that $\widehat{\text{secrecy}}(Init, n) : P$ guarantees secrecy of every bound name n in P. It follows that $\widehat{\text{secrecy}}(Init, n) : (s{:}\,\nu\, n.Q \mid t{:}\,\nu\, n.R)$ implies $\widehat{\text{secrecy}}(Init, n) : (s{:}\,\nu\, n.Q \mid t{:}\,\nu\, m.(R\{^{m}/_{n}\}) \wedge \widehat{\text{secrecy}}(Init, m) : (s{:}\,\nu\, n.Q \mid t{:}\,\nu\, m.(R\{^{m}/_{n}\}))$, for example.

Secrecy of bound names is not new. ProVerif can already check the secrecy of bound names, however, the corresponding theoretical definition has not been published. (Blanchet [11] has confirmed that Definition 6 corresponds to the secrecy of bound names notion used by ProVerif.) In addition, Ryan & Smyth [27, §3.1] propose a definition for secrecy of bound names in the applied pi calculus, however, their definition is restricted to bound names which do not appear under the scope of replication and we do not impose such a restriction. The false attacks we encounter when analysing secrecy of free names similarly occur when analysing secrecy of bound names, for instance, ProVerif cannot prove $\widehat{\text{secrecy}}(\{c\}, s) : \{c\}, \{0 : !\nu\, s.\text{P1}\}$. In the remainder of this section we overcome this problem.

4.1 A Proof Technique for Secrecy of Bound Names

It follows from our semantics and the definition of bound secrecy that: if $t{:}\,!\nu\, n.P$ does not preserve the secrecy of n, then there exists an instance of $\nu\, n.P$ that leaks n, when running in parallel with $!\nu\, n.P$. We now show that it is sufficient to rename n with some fresh name m in an instance of $\nu\, n.P$ and consider secrecy of the m in $t{:}\,\nu\, m.(P\{^{m}/_{n}\}) \mid !\nu\, n.P)$.

Theorem 2. *Given a name s, sets of names E and $Init$, process P and multiset of processes $\mathcal{P}$ such that $s \notin \mathrm{bn}(P) \cup \mathrm{bn}(\mathcal{P})$ and $\mathrm{fn}(\mathcal{P}) \cup (\mathrm{fn}(P) \setminus \{s\}) \subseteq E$, we have, for all fresh names s', that:*

$$\widehat{\mathrm{secrecy}}(Init, s) : E, \mathcal{P} \cup \{t : !\nu\, s.P\} \iff \widehat{\mathrm{secrecy}}(Init, s') : E, \mathcal{P} \cup \{t : !\nu\, s.P, t : \nu\, s'.(P\{^{s'}/_{s}\})\}$$

The proof of Theorem 1 appears in the long version of this paper.

4.2 Secrecy of Bound and Free Names Coincide

Secrecy of bound and free names coincide when the secret is not under replication.

Proposition 1. *Given a name s, sets of names E and $Init$, process P and multiset of processes $\mathcal{P}$ such that $s \notin \mathrm{bn}(P) \cup \mathrm{bn}(\mathcal{P}) \cup \mathrm{fn}(\mathcal{P}) \cup Init \cup E$ and $\mathrm{fn}(\mathcal{P}) \cup (\mathrm{fn}(P) \setminus \{s\}) \subseteq E$, we have:*

$$\widehat{\mathrm{secrecy}}(Init, s) : E, \mathcal{P} \cup \{t : \nu\, s.P\} \iff \mathrm{secrecy}(Init, s) : E \cup \{s\}, \mathcal{P} \cup \{t : P\}$$

Proof sketch. We note that $E, \mathcal{P} \cup \{t : \nu\, s.P\} \to E \cup \{s'\}, \mathcal{P} \cup \{t : P\{^{s'}/_{s}\}\}$, for a fresh name s' and that $\mathrm{secrecy}(Init, s) : E \cup \{s\}, \mathcal{P} \cup \{t : P\}$ iff $\mathrm{secrecy}(Init, s') : E \cup \{s'\}, \mathcal{P} \cup \{t : P\{^{s'}/_{s}\}\}$. Therefore, given a trace that causes free secrecy to fail to hold for the R.H.S we can add the new name declaration to produce a trace that causes secrecy to fail for the L.H.S. Conversely, given a trace that causes secrecy to fail for the L.H.S., removing the new name declaration will give us a trace that causes secrecy to fail for the R.H.S. □

A similar equivalence does not hold when the secret is under replication, as the following example demonstrates:

Example 2. For instance, suppose $P = \overline{a}\langle s\rangle \mid a(x).a(y).\mathrm{if}\ x = y\ \mathrm{then}\ \overline{c}\langle s\rangle$ and $Init = \{c\}$, we have $\widehat{\mathrm{secrecy}}(Init, s) : (0 : !\nu\, s.P)$ but not $\mathrm{secrecy}(Init, s) : (0 : !P)$, since:

$$\begin{aligned}
E, \{0 : !P, 0 : c(z)\} &\to\to E, \{0 : !P, 0 : P, 0 : P, 0 : c(z)\} && \text{by (Red Repl)}\\
&\to\to E, \{0 : !P, 0 : \overline{a}\langle s\rangle, 0 : \overline{a}\langle s\rangle, Q, Q, 0 : c(z)\} && \text{by (Red Par)}\\
&\to\to E, \mathcal{P} \cup \{0 : \mathrm{if}\ s = s\ \mathrm{then}\ \overline{c}\langle s\rangle, 0 : c(z)\} && \text{by (Red I/O)}\\
&\to E, \mathcal{P} \cup \{0 : \overline{c}\langle s\rangle, 0 : c(z)\} && \text{by (Red Destr 1)}\\
&\to E, \mathcal{P} \cup \{0 : 0, 0 : 0\{^{s}/_{z}\}\} && \text{by (Red I/O)}
\end{aligned}$$

where $E = \{a, c, s\}$, $Q = 0 : a(x).a(y).\mathrm{if}\ x = y\ \mathrm{then}\ \overline{c}\langle s\rangle$ and $\mathcal{P} = \{0 : !P, 0, 0, Q\}$.

4.3 Automated Reasoning without False Attacks

The following corollary allows us to reduce the false attacks encountered when analysing secrecy of bound names.

Corollary 1. *Given a name s, sets of names E and $Init$, and a multiset of processes $\mathcal{P} = \{0 : !\nu\, s.L, 0 : !L_1, \ldots, 0 : !L_m\}$, such that $s \notin \mathrm{bn}(L) \cup \bigcup_{1 \leq i \leq n}(\mathrm{fn}(L_i) \cup \mathrm{bn}(L_i))$ and $\mathrm{fn}(\mathcal{P}) \subseteq E$, we have for all fresh names s' and $\mathcal{Q} \in \Delta(Init, \mathcal{P} \cup \{L\{^{s'}/_{s}\}\})$ that:*

$$\widehat{\mathrm{secrecy}}(Init, s) : E, \mathcal{P} \Leftrightarrow \mathrm{secrecy}(Init, s') : E \cup \{s'\}, \mathcal{Q}$$

Proof. We have:

$$\begin{aligned}
&\widehat{\mathrm{secrecy}}(Init, s) : E, \mathcal{P} \\
&\quad \Leftrightarrow \widehat{\mathrm{secrecy}}(Init, s') : E, \{0 : \nu\, s'.(L\{^{s'}/_{s}\})\} \cup \mathcal{P} && \text{by Theorem 2} \\
&\quad \Leftrightarrow \mathrm{secrecy}(Init, s') : E \cup \{s'\}, \{0 : L\{^{s'}/_{s}\}\} \cup \mathcal{P} && \text{by Proposition 1} \\
&\quad \Leftrightarrow \mathrm{secrecy}(Init, s') : E \cup \{s'\}, \mathcal{Q} && \text{by Theorem 1}
\end{aligned}$$

□

We demonstrate an application of Corollary 1 by evaluating $\widehat{\mathrm{secrecy}}(\{c\}, s) : \{c\}, \{!\nu\, s.\mathrm{P1}\}$:

Example 3. Witness that $\nu\, s.\mathrm{P1} = C[c(x).\overline{c}\langle n\rangle.\text{if } x = n \text{ then } \overline{c}\langle s\rangle]$, where $C = \nu\, n.\_$, but there is no other context C, process L and terms M and x such that $\nu\, s.\mathrm{P1} = C[M(x).L]$. In addition, there is no context C, process L and terms M, N such that $\nu\, s.\mathrm{P1} = C[\overline{M}\langle N\rangle.L] \wedge \mathrm{fn}(M) \cap \{c\} = \emptyset$. It follows that $\delta(\{c\}, \nu\, s.0 : \mathrm{P1}) = \{0 : !\nu\, s.\mathrm{P1}, 0 : !\nu\, s.1 : c(x).\overline{c}\langle n\rangle.\text{if } x = n \text{ then } \overline{c}\langle s\rangle\}$. Let $\mathcal{Q} = \delta(\{c\}, 0 : \nu\, s.\mathrm{P1}) \cup \{\mathrm{P3}\}$, where P3 is defined as follows:

$$0 : \nu\, n.c(x).1 : \overline{c}\langle n\rangle.\text{if } x = n \text{ then } \overline{c}\langle s'\rangle \qquad \text{(P3)}$$

Since $\mathrm{P1}\{^{s'}/_{s}\} = C[\overline{c}\langle n\rangle.\text{if } x = n \text{ then } \overline{c}\langle s'\rangle]$, where $C = \nu\, n.c(x).\_$, we have $\mathrm{P3} \in \delta(\emptyset, 0 : \mathrm{P1}\{^{s'}/_{s}\})$ and it follows that $\mathcal{Q} \in \Delta(\{c\}, \{0 : \mathrm{P1}\{^{s'}/_{s}\}, 0 : !\nu\, s.\mathrm{P1}\})$. We have $\widehat{\mathrm{secrecy}}(\{c\}, s) : \{c\}, \{0 : !\nu\, s.\mathrm{P1}\} \Leftrightarrow \mathrm{secrecy}(\{c\}, s') : \{c, s'\}, \mathcal{Q}$ by Corollary 1 and, since ProVerif can prove $\mathrm{secrecy}(\{c\}, s) : \mathcal{Q}$, we have the desired result $\widehat{\mathrm{secrecy}}(\{c\}, s) : \{c\}, \{!\nu\, s.\mathrm{P1}\}$.

5 Case Study I: Needham-Schroeder Protocol

For our first case study, we compose the Needham-Schroeder protocol [26] with a symmetric encryption scheme to derive a secure channel.

Needham-Schroeder protocol. The Needham-Schroeder protocol [26] is intended to generate a session key shared between two participants. We assume the participants are Alice and Bob, to prevent Lowe's man-in-the-middle attack[2] [24]. The protocol proceeds as follows. First, Alice generates a nonce and outputs

[2] Lowe's attack works as follows: an attacker engages Alice in a session of the protocol and impersonates Alice to Bob in a parallel session. The attack can be thwarted by assuming that Alice will only run the protocol with Bob, rather than any other principal.

the nonce encrypted with Bob's public key. Secondly, Bob decrypts Alice's ciphertext using his private key to recover Alice's nonce, generates a nonce, and outputs the pair of nonces encrypted with Alice's public key. Finally, Alice decrypts Bob's ciphertext using her private key to recover the pair of nonces and outputs Bob's nonce encrypted with Bob's public key. The rationale behind the protocol is that: since only Bob can recover Alice's nonce, only he can output the encrypted pair of nonces, moreover, since only Alice can recover Bob's nonce, only she can output his encrypted nonce; it follows that the two nonces are only known to Alice and Bob.

Symmetric encryption scheme. Symmetric encryption enables a secret to be shared between two participants. We consider the following symmetric encryption scheme. Given an identifier, key, and secret as input, the initiator outputs the identifier paired with the secret encrypted with the key. Upon receipt of such a pair, the interlocutor uses the key associated with the initiator's identifier to decrypt the ciphertext. It follows from our description that the initiator's secret can only be known by the key holders.

We compose the Needham-Schroeder protocol and symmetric encryption scheme to derive a secure channel[3]. The Needham-Schroeder protocol is used to generate a pair of nonces and these nonces are used by the symmetric encryption scheme as follows: Alice's nonce is used as the identifier and Bob's nonce is used as the key. Intuitively, the composition ensures that Alice's secret is known only to Alice and Bob, because Bob's nonce is only known to Alice and Bob.

5.1 Applied pi Calculus Model

We construct a signature Σ to capture the primitives modelling cryptographic operators and constants: $\Sigma = \{\mathsf{fst}, \mathsf{snd}, \mathsf{pk}, \mathsf{pair}, \ \mathsf{adec}, \mathsf{aenc}, \mathsf{sdec}, \mathsf{senc}\}$, where $\mathsf{fst}, \mathsf{snd}, \mathsf{pk}$ are unary functions and $\mathsf{adec}, \mathsf{aenc}, \mathsf{sdec}, \ \mathsf{senc}$ are binary functions. We equip the signature with the following rewrite rules:

$$\{\mathsf{fst}(\mathsf{pair}(x,y)) \to x, \ \mathsf{snd}(\mathsf{pair}(x,y)) \to y, \\ \mathsf{adec}(x, \mathsf{aenc}(\mathsf{pk}(x), y)) \to y, \ \mathsf{sdec}(x, \mathsf{senc}(x,y)) \to y\}.$$

Our signature and associated rewrite rules allow us to model: asymmetric encryption, pairing, and symmetric encryption.

We define the participants, Alice and Bob, in our composition of the Needham-Schroeder protocol with a symmetric encryption scheme (Figure 4). Hence, the complete composition is modelled by the configuration $E_{\mathsf{NSL}}, \mathcal{P}_{\mathsf{NSL}} = \{c, k_A, k_B\}$, $\{!\nu\, s.A, !B, !\overline{c}\langle\mathsf{pk}(k_A)\rangle, !\overline{c}\langle\mathsf{pk}(k_B)\rangle\}$ (which are implicitly assumed to be running in phase 0).

[3] This example was inspired by discussion with Blanchet & Cortier [14].

Fig. 4. Processes modelling Alice and Bob in the Needham-Schroeder protocol

$$
\begin{aligned}
A = {} & \nu\, n_A.\\
& \overline{c}\langle \mathsf{aenc}(\mathsf{pk}(k_B), n_A)\rangle.\\
& c(x_{\mathsf{ciph}}).\\
& \text{let } x_{\mathsf{pair}} = \mathsf{adec}(k_A, x_{\mathsf{ciph}}) \text{ in}\\
& \text{let } y_{\mathsf{nonce}} = \mathsf{fst}(x_{\mathsf{pair}}) \text{ in}\\
& \text{if } y_{\mathsf{nonce}} = n_A \text{ then}\\
& \text{let } x_{\mathsf{nonce}} = \mathsf{snd}(x_{\mathsf{pair}}) \text{ in}\\
& \overline{c}\langle \mathsf{aenc}(\mathsf{pk}(k_B), x_{\mathsf{nonce}})\rangle.\\
& (* \text{ end key exchange } *)\\
& \overline{c}\langle \mathsf{pair}(n_A, \mathsf{senc}(x_{\mathsf{nonce}}, s))\rangle
\end{aligned}
\qquad
\begin{aligned}
B = {} & c(y_{\mathsf{ciph}}).\\
& \text{let } y_{\mathsf{nonce}} = \mathsf{adec}(k_B, y_{\mathsf{ciph}}) \text{ in}\\
& \nu\, n_B.\\
& \overline{c}\langle \mathsf{aenc}(\mathsf{pk}(k_A), \mathsf{pair}(y_{\mathsf{nonce}}, n_B))\rangle.\\
& c(y'_{\mathsf{ciph}}).\\
& \text{let } x_{\mathsf{nonce}} = \mathsf{adec}(k_B, y'_{\mathsf{ciph}}) \text{ in}\\
& \text{if } x_{\mathsf{nonce}} = n_B \text{ then}\\
& c(y_{\mathsf{pair}}).\\
& \text{let } y'_{\mathsf{nonce}} = \mathrm{fst}(y_{\mathsf{pair}}) \text{ in}\\
& \text{if } y_{\mathsf{nonce}} = y'_{\mathsf{nonce}} \text{ then}\\
& \text{let } y''_{\mathsf{ciph}} = \mathsf{snd}(y_{\mathsf{pair}}) \text{ in}\\
& \text{let } y_{\mathsf{secret}} = \mathsf{sdec}(n_B, y''_{\mathsf{ciph}}) \text{ in } 0
\end{aligned}
$$

5.2 Analysis

We would like to analyse $\widehat{\mathsf{secrecy}}(\{c\}, s) : E_{\mathsf{NSL}}, \mathcal{P}_{\mathsf{NSL}}$ using ProVerif. However, the Horn clauses generated by Blanchet's translation of $\mathcal{P}_{\mathsf{NSL}}$ result in Horn clauses which model the process $\mathcal{P}_{\mathsf{NSL}}$ with a replication after the name restriction $\nu\, n_A$. Unfortunately, secrecy does not hold in this process, because the attacker can learn n_A during one run of the protocol and simulate Bob during a second run to learn Alice's secret s. Indeed, ProVerif finds such an attack. However, this is a false attack, because the protocol states that Alice should use a fresh nonce for every session of the protocol. This false attack can be avoided using our results.

By Corollary 1, to prove $\widehat{\mathsf{secrecy}}(\{c\}, s) : E_{\mathsf{NSL}}, \mathcal{P}_{\mathsf{NSL}}$ it is sufficient to prove $\mathrm{secrecy}(\mathit{Init}, s') : E \cup \{s'\}, \mathcal{Q}$, where $\mathcal{Q} \in \Delta(\{c\}, \mathcal{P}_{\mathsf{NSL}} \cup \{A\{^{s'}/_{s}\}\})$. Let us construct a suitable $\mathcal{Q}$. By definition of Δ (Definition 4), we have $\mathcal{Q} = \{L\} \cup \bigcup_{P \in \mathcal{P}_{\mathsf{NSL}}} \{!R \mid R \in \delta(\mathit{Init}, P)\}$ for some $L \in \delta(\emptyset, A\{^{s'}/_{s}\})$. Since $\mathcal{P}_{\mathsf{NSL}}$ does not contain any private channels, for all contexts C, processes L and terms M and N such that $\mathrm{fv}(M) \cap \{c\} = \emptyset$, we have $\nu\, s.A \neq C[\overline{M}\langle N\rangle.L']$, $B \neq C[\overline{M}\langle N\rangle.L']$, $\overline{c}\langle \mathsf{pk}(k_A)\rangle \neq C[\overline{M}\langle N\rangle.L']$, and $\overline{c}\langle \mathsf{pk}(k_B)\rangle \neq C[\overline{M}\langle N\rangle.L']$. Moreover, since $\overline{c}\langle \mathsf{pk}(k_A)\rangle$ and $\overline{c}\langle \mathsf{pk}(k_B)\rangle$ do not contain inputs, we have $\{!R \mid R \in \delta(\mathit{Init}, \overline{c}\langle \mathsf{pk}(k_A)\rangle)\} = \{!\overline{c}\langle \mathsf{pk}(k_A)\rangle\}$ and, similarly, $\{!R \mid R \in \delta(\mathit{Init}, \overline{c}\langle \mathsf{pk}(k_B)\rangle)\} = \{!\overline{c}\langle \mathsf{pk}(k_B)\rangle\}$. The set $\{!R \mid R \in \delta(\mathit{Init}, \nu\, s.A)\}$ contains $!\nu\, s.A$ and a modified version of $!\nu\, s.A$, namely, $!\nu\, s.A$ with a phase inserted before the input. The set $\{!R \mid R \in \delta(\mathit{Init}, B)\}$ contains $!B$ and three modified versions of $!B$, namely, $!B$ with a phase before each of the three inputs. We have a concrete definition of $\bigcup_{P \in \mathcal{P}_{\mathsf{NSL}}} \{!R \mid R \in \delta(\mathit{Init}, P)\}$ and we let $L \in \delta(\emptyset, A\{^{s'}/_{s}\})$ be $A\{^{s'}/_{s}\}$ with a phase inserted before the penultimate output. ProVerif can automatically verify that $\mathrm{secrecy}(\mathit{Init}, s') : E \cup \{s'\}, \mathcal{Q}$ holds (verification takes less than one second using ProVerif 1.86pl4 on Ubuntu 12.04.3 with 3.60GHz Intel Xeon E5-1620 and 8GB memory).

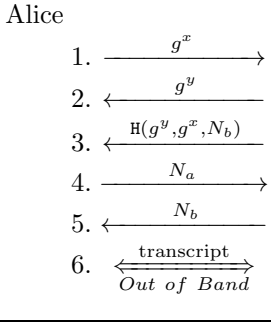
6 Case Study II: Bluetooth Simple Pairing

The Bluetooth Simple Pairing protocol [22,23] extends the (elliptic curve) Diffie-Hellman protocol to provide authenticated key exchange. There are a number of variations of this protocol, depending on the capabilities of the devices being paired, here we look at the "Numeric Comparison Protocol" that aims to securely pair devices that are capable of displaying a short number on a screen, and receiving an input from the user. The displays of the participants' devices provide a low-bandwidth, authenticated "out of band" channel, which is assumed to be untappable by the attacker. The bandwidth constraint of this out of band channel makes BlueTooth Pairing non-trivial.

The protocol (Figure 5) proceeds as follows. Alice and Bob establish a Diffie-Hellman key (Steps 1 & 2). Bob generates a nonce and outputs a commitment to his nonce (Step 3), Alice outputs a nonce (Step 4), and Bob reveals his nonce (Step 5). Alice and Bob each establish their transcript of values g^x, g^y, N_a, N_b and check that the first few characters of their transcripts – which we write as $\texttt{short}(g^x, g^y, N_a, N_b)$ – match using their out of band channel (Step 6). Alice and Bob can use the Diffie-Hellman key g^{xy}.

The rationale behind the protocol is that: checking the transcripts on an authenticated channel guarantees the key is shared between Alice and Bob. However, Alice and Bob are only required to check a few characters of the transcript, because the authenticated channel is low-bandwidth, which limits the number of characters that can be checked. Formally, we derive the low-entropy string of the characters that should be checked by applying function short to the values g^x, g^y, N_a, N_b.

Fig. 5. BlueTooth Simple Pairing



Collision attacks. String $\texttt{short}(g^x, g^y, N_a, N_b)$ is low-entropy, therefore, unlike computing a hash on g^x, g^y, N_a, N_b, it is computationally feasible to find a collision. The following example demonstrates collision attacks.

Example 4. Consider a variant of BlueTooth Simple Pairing, without step 3. In this protocol, the attacker can replace Alice's message g^x with g^z and Bob's message N_b with N_e such that $\texttt{short}(g^z, g^y, N_a, N_b) = \texttt{short}(g^x, g^z, N_a, N_e)$. (This would require the attacker to make many guesses for the nonce N_e.) Given that Alice and Bob would now see the same low-entropy string, the attacker can impersonate Alice.

Such an attack is prevented in BlueTooth Simple Pairing, by forcing both participants to commit to their nonces before they see the other participant's nonce. Alice must send her nonce first, and Bob must send a hash of this nonce before he sees Alice's Nonce. Alice will check that this nonce matches the hash

between steps 5 and 6 of the protocol. This ensures that the attacker never has the opportunity to launch an impersonation attack.

We remark that collision attacks are conceptually different from guessing attacks against weak secrets [6,16], which occur when an attacker gets enough information from a protocol to verify a guess of a value, which is low entropy. The kind of brute force attack we describe here occurs when an attacker has some control over the inputs to a high-entropy function and needs to force the short code based on the output of that function to equal a certain value. In particular, if an attacker controls one input and knows the values of all the other inputs, the attacker can generated a large number of possible inputs and, by brute force, find an input of their own that makes the output of the function produce a value that matches any short code they wish.

6.1 Applied pi Calculus Model

Our model assumes that an attacker can generate sufficiently many transcripts such that distinct transcripts share the same first few characters. That is, given $\mathtt{short}(M_1, M_2, M_3, M_4)$, the adversary can compute $\mathtt{short}(N_1, N_2, N_3, N_4)$ such that $\mathtt{short}(M_1, M_2, M_3, M_4) = \mathtt{short}(N_1, N_2, N_3, N_4)$. Function `bruteforce` captures this. Moreover, we supplement `short` with `shortb` to enable automated analysis.

We construct a signature $\Sigma = \{\mathtt{g},\mathtt{f},\mathtt{sdec},\mathtt{senc},\mathtt{H},\mathtt{bruteforce},\mathtt{short},\mathtt{shortb}\}$, where `g` is a unary function, `sdec`, `senc` and `f` are binary functions, `H` is a ternary function, and `bruteforce`, `short` and `shortb` take four arguments. Function `H` represents a hash function, `sdec` and `senc` capture symmetric encryption, i.e., $\mathtt{def(short)} = \{\mathsf{sdec}(x, \mathsf{senc}(x, y)) \to y\}$, and the purpose of the remaining functions is explained below.

Diffie-Hellman key agreement is modelled in the standard fashion [10, §9.1] using functions `f` and `g` and the following equation:

$$\mathtt{f}(x, \mathtt{g}(y)) = \mathtt{f}(y, \mathtt{g}(x))$$

which allows us to capture $(g^x)^y = (g^y)^x$. (A more general setting is beyond the scope of [10, §9.1].)

Function `shortb` is used to model a low-entropy string derived from its input. For instance, $\mathtt{shortb}(g^x, g^y, N_A, N_B)$ represents a low-entropy string derived from terms g^x, g^y, N_A, and N_B. To capture collision attacks, function `shortb` is not used directly (that is, we do not use `shortb` in processes), instead, destructor `short` is used and we map occurrences of `short` to `shortb` using the following rewrite rule:

$$\mathtt{short}(w, x, y, z) \to \mathtt{shortb}(w, x, y, z)$$

Moreover, we add the following rewrite rules to capture collisions:

$$\begin{aligned}
&\mathtt{short}(w, x, y, \mathtt{bruteforce}(w, x, y, \mathtt{shortb}(\hat{w}, \hat{x}, \hat{y}, \hat{z}))) \to \mathtt{shortb}(\hat{w}, \hat{x}, \hat{y}, \hat{z})\\
&\mathtt{short}(w, x, \mathtt{bruteforce}(w, x, z, \mathtt{shortb}(\hat{w}, \hat{x}, \hat{y}, \hat{z})), z) \to \mathtt{shortb}(\hat{w}, \hat{x}, \hat{y}, \hat{z})\\
&\mathtt{short}(w, \mathtt{bruteforce}(w, y, z, \mathtt{shortb}(\hat{w}, \hat{x}, \hat{y}, \hat{z})), y, z) \to \mathtt{shortb}(\hat{w}, \hat{x}, \hat{y}, \hat{z})\\
&\mathtt{short}(\mathtt{bruteforce}(x, y, z, \mathtt{shortb}(\hat{w}, \hat{x}, \hat{y}, \hat{z})), x, y, z) \to \mathtt{shortb}(\hat{w}, \hat{x}, \hat{y}, \hat{z})
\end{aligned}$$

It follows from our rewrite rules that

$$\texttt{short}(M_1, M_2, M_3, M_4) \rightarrow \texttt{shortb}(M_1, M_2, M_3, M_4)$$

and

$$\texttt{short}(N_1, N_2, N_3, \texttt{bruteforce}(N_1, N_2, N_3, \texttt{shortb}(M_1, M_2, M_3, M_4))) \rightarrow \texttt{shortb}(M_1, M_2, M_3, M_4))$$

i.e., terms $\texttt{short}(M_1, M_2, M_3, M_4)$ and $\texttt{short}(N_1, N_2, N_3, \texttt{bruteforce}(N_1, N_2, N_3, \texttt{shortb}(M_1, M_2, M_3, M_4)))$ collide.

The ProVerif source of this, and all our other examples, are available from the following URL: `http://www.cs.bham.ac.uk/~tpc/projects/falseattacks`.

6.2 Analysis

We analyse this protocol by adding in the exchange of a value at the end of the protocol using the key g^{xy} and testing for the secrecy of this value. Running our model in ProVerif we find that it results in a false attack, due to the commitment problem that was discussed above. The tool suggests that the attacker can send a dummy commitment value and then observe nonces N_a and N_b. The attacker can then use a brute forced value based on N_b and start the protocol again with a commitment to this new brute forced value, however, in this second run of the protocol Bob would be using a different N_b value and so the attack is a false one.

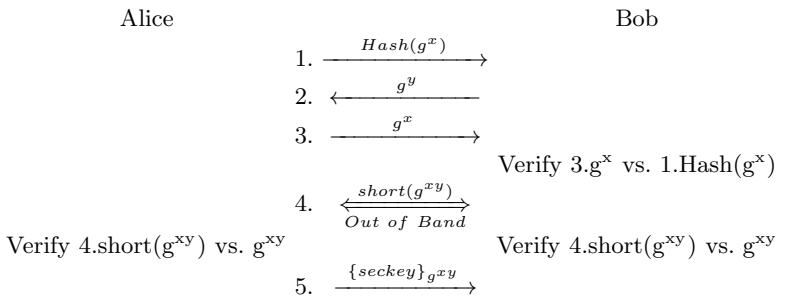
Applying Theorem 1 to transform the BlueTooth Protocol results in a model (available from the aforementioned URL) for which ProVerif can verify that the secrecy of *seckey* in less than a second, and so Theorem 1 then tells us that, in spite of the false attack, the original protocol is also secure. If we remove the check of the commitment sent in step 3 of the protocol, then ProVerif finds the attack in which the attacker performs a man in the middle attack using the `bruteforce` function to find a value which matches the short code sent between Alice and Bob.

6.3 Case Study III: The Simplified Simple Pairing Protocol

Our final case study proposes a simplified version of the Bluetooth Simple Pairing protocol (Figure 6). Our new protocol merges the tasks of the Diffie-Hellman exponents and the nonces. The out of band channel is used for confirmations of a short code based on the Diffie-Hellman key. We note that in the specification of the Bluetooth Simple Pairing protocol [23] the Diffie-Hellman steps of the protocol are presented as a method of stopping eavesdropping attacks, and the steps using the nonce values are presented quite separately as a method of stopping active attacks; the contribution of our protocol is to show that these steps can be merged. We also note that in the Bluetooth Simple Pairing protocol the channel is protected by the entropy of the secret Diffie-Hellman values and the

nonces, whereas in our new protocol it is just protected by the entropy of the secret Diffie-Hellman values. Therefore, for our protocol to be secure, the secret Diffie-Hellman values must be high-entropy and fresh; this is not currently an explicit requirement stated in the Bluetooth specification.

Fig. 6. SSP: Simplified Simple Pairing



Our new protocol starts with Alice sending a commitment to a particular Diffie-Hellman exponent. Bob then sends his exponent and Alice replies by sending her's. Bob must check that the exponent he receives matches the commitment to avoid the brute force attack described above. Alice and Bob then compare the transcript of the key on their out of bound channel. The nonces used in the Bluetooth Simple Pairing protocol are no longer needed, as the freshness guarantees are now provided by the values x and y.

A brute force attack may be tried against the short string in this protocol. To allow this in our model, we adapt the rules for `bruteforce` and `short` to work against Diffie-Hellman exponents. In this case, the `short` function is applied to the `f` function i.e., `short(f(s,g(t)))` represents the short string generated from the Diffie-Hellman key `f(s,g(t))`.

Given `g(t)` and `M`, the attacker can derive a term `N` such that `short(f(N,g(t))) = short(M)`, and we write this `N` value as `bruteforce(g(t),M)` and capture the desired relation using the following rewrite rules:

$$\begin{aligned} \texttt{short}(x) &\rightarrow \texttt{shortb}(x) \\ \texttt{short}(\texttt{f}(\texttt{bruteforce}(x, \texttt{shortb}(y)), x)) &\rightarrow \texttt{shortb}(y) \\ \texttt{short}(y, \texttt{f}(\texttt{bruteforce}(y, \texttt{shortb}(x)))) &\rightarrow \texttt{shortb}(x) \end{aligned}$$

It follows from the above rules that given `g(t)` and `f(s,g(t))`, the attacker can derive `bruteforce(g(t),short(f(s,g(t))))` such that `short(f(bruteforce(g(t), short(f(s,g(t)))),g(t))) = short(f(s,g(t)))`, thereby modelling a brute force attack against the Diffie-Hellman exponents.

Analysing this scheme using ProVerif results in the discovery of a false attack, whereas Theorem 1 allows us to verify the secrecy of *seckey*.

7 Conclusion

We have shown how false atacks can be avoided when analysing secrecy with ProVerif. Our method works by inserting phases into processes such that they enforce an ordering on Horn clauses. We demonstrate the applicability of our methodology by analysing BlueTooth Pairing. This case study leads use to develop theory to enable the analysis of protocols that are vulnerable to impersonation attacks. Finally, we show that BlueTooth Pairing can be simplified and show that the simplified scheme satisfies the same secrecy objectives.

Acknowledgements. We thank Bruno Blanchet for insightful discussions which have influenced this paper. This work was performed in part at INRIA with support from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC project *CRYSP* (259639).

References

1. Abadi, M., Blanchet, B., Fournet, C.: Just Fast Keying in the Pi Calculus. ACM Transactions on Information and System Security 10(3) (2007)
2. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL 2001: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 104–115. ACM Press (2001)
3. Allamigeon, X., Blanchet, B.: Reconstruction of Attacks against Cryptographic Protocols. In: CSFW 2005: 18th Computer Security Foundations Workshop, pp. 140–154. IEEE Computer Society (2005)
4. Arapinis, M., Cortier, V., Kremer, S., Ryan, M.: Practical everlasting privacy. In: Basin, D., Mitchell, J.C. (eds.) POST 2013. LNCS, vol. 7796, pp. 21–40. Springer, Heidelberg (2013)
5. Backes, M., Hriţcu, C., Maffei, M.: Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-calculus. In: CSF 2008: 21st IEEE Computer Security Foundations Symposium, pp. 195–209. IEEE Computer Society (2008)
6. Baudet, M.: Deciding security of protocols against off-line guessing attacks. In: Proc. 12th ACM Conference on Computer and Communications Security (CCS 2005), pp. 16–25. ACM Press (2005)
7. Baudet, M.: Sécurité des protocoles cryptographiques: Aspects logiques et calculatoires. PhD thesis, Laboratoire Spécification et Vérification, ENS Cachan, France (2007)
8. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: CSFW 2001: 14th IEEE Computer Security Foundations Workshop, pp. 82–96. IEEE Computer Society (2001)
9. Blanchet, B.: From Secrecy to Authenticity in Security Protocols. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 342–359. Springer, Heidelberg (2002)
10. Blanchet, B.: Automatic Verification of Correspondences for Security Protocols. Journal of Computer Security 17(4), 363–434 (2009)
11. Blanchet, B.: Private email communication (November 12, 2012)

12. Blanchet, B.: Security Protocol Verification: Symbolic and Computational Models. In: Degano, P., Guttman, J.D. (eds.) Principles of Security and Trust. LNCS, vol. 7215, pp. 3–29. Springer, Heidelberg (2012)
13. 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)
14. Blanchet, B., Cortier, V.: Private email communication (November 13, 2012)
15. Blanchet, B., Smyth, B.: ProVerif: Automatic Cryptographic Protocol Verifier User Manual & Tutorial (2011), `http://www.proverif.ens.fr/`
16. Blanchet, B., Smyth, B., Cheval, V.: Proverif 1.88: Automatic cryptographic protocol verifier, user manual and tutorial (2013)
17. Chang, R., Shmatikov, V.: Formal analysis of authentication in bluetooth device pairing. In: Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (2007)
18. Chen, L., Ryan, M.: Attack, Solution and Verification for Shared Authorisation Data in TCG TPM. In: Degano, P., Guttman, J.D. (eds.) FAST 2009. LNCS, vol. 5983, pp. 201–216. Springer, Heidelberg (2010)
19. Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: Formal analysis of protocols based on tpm state registers. In: CSF 2011: 24th IEEE Computer Security Foundations Symposium, pp. 66–80. IEEE (2011)
20. Delaune, S., Ryan, M.D., Smyth, B.: Automatic verification of privacy properties in the applied pi-calculus. In: Karabulut, Y., Mitchell, J., Herrmann, P., Jensen, C.D. (eds.) IFIPTM 2008: 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security. IFIP, vol. 263, pp. 263–278. Springer, Heidelberg (2008)
21. Durgin, N.A., Lincoln, P., Mitchell, J.C., Scedrov, A.: Multiset rewriting and the complexity of bounded security protocols. Journal of Computer Security 12(2), 247–311 (2004)
22. Bluetooth Special Interest Group. Specification of the bluetooth system (2001)
23. Bluetooth Special Interest Group. Simple pairing whitepaper (2006)
24. Breaking, G.L.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
25. Meadows, C.: Open Issues in Formal Methods for Cryptographic Protocol Analysis. In: Gorodetski, V.I., Skormin, V.A., Popyack, L.J. (eds.) MMM-ACNS 2001. LNCS, vol. 2052, p. 21. Springer, Heidelberg (2001)
26. Needham, R.M., Schroeder, M.D.: Using Encryption for Authentication in Large Networks of Computers. Communications of the ACM 21(12), 993–999 (1978)
27. Ryan, M.D., Smyth, B.: Applied pi calculus. In: Cortier, V., Kremer, S. (eds.) Formal Models and Techniques for Analyzing Security Protocols, ch. 6. IOS Press (2011)
28. Smyth, B., Ryan, M.D., Chen, L.: Formal analysis of privacy in Direct Anonymous Attestation schemes (2012)
29. Zhao, F., Hanatani, Y., Komano, Y., Smyth, B., Ito, S., Kambayashi, T.: Secure Authenticated Key Exchange with Revocation for Smart Grid. In: ISGT 2012: 3rd IEEE Power & Energy Society Conference on Innovative Smart Grid Technologies, pp. 1–8 (2012)