Applied pi calculus

Mark D. RYAN and Ben SMYTH
School of Computer Science,
University of Birmingham,
United Kingdom
e-mail: M.D.Ryan@cs.bham.ac.uk, research@bensmyth.com

Abstract. The applied pi calculus is a language for modelling security protocols. It is an extension of the pi calculus, a language for studying concurrency and process interaction. This chapter presents the applied pi calculus in a tutorial style. It describes reachability, correspondence, and observational equivalence properties, with examples showing how to model secrecy, authentication, and privacy aspects of protocols.

Keywords: Cryptographic protocols, protocol verification, formal methods, reachability, correspondence properties, observational equivalence, tutorial.

1. Introduction

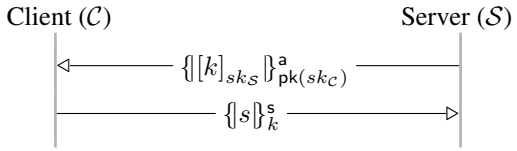
The applied pi calculus [8] is a language for describing and analysing security protocols. It provides an intuitive process syntax for detailing the actions of the participants in a protocol, emphasising their communication. The syntax is coupled with a formal semantics to allow reasoning about protocols. The language is based on the pi calculus with the addition of a rich term algebra to enable modelling of the cryptographic operations used by security protocols. A wide variety of cryptographic primitives can be abstractly modelled by means of an equational theory. The calculus allows one to express several types of security goal, and to analyse whether the protocol meets its goal or not. This analysis can sometimes be performed automatically, using the ProVerif software tool [23,21,25].

The applied pi calculus has been used to model security protocols in a variety of areas. The following examples are not exhaustive:

- Certified email [4];
- Privacy properties [38,34,12,40], and election-verifiability properties [39] in electronic voting;
- Authorisation protocols [27,43], and attestation protocols [46,13] in trusted computing;
- Interoperability of web services, where a compiler [17] converts descriptions in the F# language [47] into applied pi, suitable for analysis by ProVerif;
- Integrity of file systems on untrusted storage [24];
- Authentication protocols and key agreement [5].

1.1. Handshake protocol

We recall the naïve Handshake protocol (Chapter *"Introduction"*) between a client $\mathcal{C}$ and server $\mathcal{S}$. It is assumed that each of them has a public/private key pair, and that the client knows the server's public key $\mathsf{pk}(sk_{\mathcal{S}})$. The aim of the protocol is to establish a secret symmetric key k, enabling the client to communicate a secret s to the server. The protocol proceeds as follows. On request from a client $\mathcal{C}$, server $\mathcal{S}$ generates a fresh session key k, signs it with her private key $sk_{\mathcal{S}}$ and encrypts it using her client's public key $\mathsf{pk}(sk_{\mathcal{C}})$. When $\mathcal{C}$ receives this message he decrypts it using his private key $sk_{\mathcal{C}}$, verifies the digital signature made by $\mathcal{S}$ using her public key $\mathsf{pk}(sk_{\mathcal{S}})$, and extracts the session key k. $\mathcal{C}$ uses this key to symmetrically encrypt the secret s and sends the encrypted message to $\mathcal{S}$. The rationale behind the protocol is that $\mathcal{C}$ receives the signature asymmetrically encrypted with his public key and hence he should be the only one able to decrypt its content. Moreover, the digital signature should ensure that $\mathcal{S}$ is the originator of the message. The protocol narration is illustrated as follows:



Note that protocol narrations (as above) are useful, but lack detail. For example, they do not specify the construction of nonces, nor do they describe any checks which should be made by the participants during the execution of the protocol. Such checks include verifying digital signatures and ensuring that encrypted messages are correctly formed. Failure of these checks typically results in the participant aborting the protocol. These details will be explicitly stated when protocols are modelled in the applied pi calculus. (For further discussion on protocol specification see [11,1].)

Informally, the three properties we would like this protocol to provide are:

1. Secrecy: The value s is known only to $\mathcal{C}$ and $\mathcal{S}$.
2. Authentication of $\mathcal{S}$: if $\mathcal{C}$ reaches the end of the protocol with session key k, then $\mathcal{S}$ proposed k for use by $\mathcal{C}$.
3. Authentication of $\mathcal{C}$: if $\mathcal{S}$ reaches the end of the protocol and believes session key k has been shared with $\mathcal{C}$, then $\mathcal{C}$ was indeed her interlocutor and has k.

The different forms of the two authentication properties arise because of the different assumptions we made about $\mathcal{C}$ and $\mathcal{S}$. Recall that $\mathcal{C}$ knows $\mathcal{S}$'s public key, and is only willing to run the protocol with $\mathcal{S}$. But $\mathcal{S}$ is willing to run the protocol with anyone.

Careful analysis reveals that the protocol does not satisfy all three of the intended properties. It is vulnerable to a *man-in-the-middle* attack. If a dishonest participant $\mathcal{M}$ starts a session with $\mathcal{S}$, then $\mathcal{M}$ is able to impersonate $\mathcal{S}$ in a subsequent session he starts with $\mathcal{C}$. At the end of the protocol $\mathcal{C}$ believes that he shares the secret s with $\mathcal{S}$, while he actually shares s with $\mathcal{M}$. (This attack is further detailed in the extended version of this chapter [44].) The protocol can easily be corrected by adding the identities of the intended participants to the data that is signed in the first message. With this correction, $\mathcal{M}$ is not able to re-use the signed key from $\mathcal{S}$ in his session with $\mathcal{C}$.

2. Applied pi calculus

The applied pi calculus [8] is a language for describing concurrent processes and their interactions. It is based on the pi calculus but is intended to be more convenient to use, and it is specifically targeted at modelling security protocols. In this respect the applied pi calculus also has similarities with the spi calculus [10]. The key difference concerns the way in which cryptographic primitives are handled. The spi calculus has a fixed set of primitives built-in (namely, symmetric and public key encryption), while the applied pi calculus allows a wide variety of more complex primitives (including, for example, non-deterministic encryption, digital signatures, and proofs of knowledge) to be defined by means of an equational theory.

2.1. Syntax and informal semantics

The calculus assumes an infinite set of names, an infinite set of variables and a signature Σ consisting of a finite set of function symbols each with an associated arity. Function symbols capture primitives used by cryptographic protocols (for example: one-way hash functions, encryption, digital signatures, and data structures such as pairing). A function symbol with arity 0 is a constant. Terms are built by applying function symbols to names, variables and other terms:

$L, M, N, T, U, V ::=$	terms
$a, b, c, \ldots, k, \ldots, m, n, \ldots, s$	name
x, y, z	variable
$g(M_1, \ldots, M_l)$	function application

where g ranges over the functions of Σ and l is the arity of g. We use metavariables u, v, w to range over both names and variables. Tuples $u_1, \ldots, u_l$ and $M_1, \ldots, M_l$ are occasionally abbreviated $\tilde{u}$ and $\tilde{M}$. We write $\{^M/_x\}$ for the *syntactic substitution* (usually just called *substitution*) that replaces the variable x with the term M. Similarly, we write $\{^m/_n\}$ for the substitution that replaces the name n with the name m. Arbitrarily large substitutions can be written as $\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$ or $\{^{\tilde{M}}/_{\tilde{x}}\}$. The letters σ and τ range over substitutions. We write $N\sigma$ for the result of applying σ to the free variables (or free names) of N. A term is ground when it does not contain variables.

We assume a type system (also known as a sort system) for terms generated by a set of base types $\mathcal{S}$, which includes the universal type Data. In addition, if ω is a type, then $\mathsf{Channel}\langle\omega\rangle$ is a type too. Formally, the set of types generated by the base types $\mathcal{S}$ is the smallest set Ω satisfying: 1) $\mathcal{S} \subseteq \Omega$; and 2) if $\omega \in \Omega$ then $\mathsf{Channel}\langle\omega\rangle \in \Omega$. Names and variables can have any type. By convention we use a, b, c for channel names, k, s as names of base type and m, n for names of any type. A channel of type $\mathsf{Channel}\langle\omega\rangle$ may communicate messages of type ω. For simplicity, function symbols can only be applied to, and return, terms of base type. We always assume that terms are well-typed and that syntactic substitutions preserve types.

The grammar for *processes* is shown below:

$P, Q, R ::=$	processes (or plain processes)
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu\, n.P$	name restriction
if $M = N$ then P else Q	conditional
$u(x).P$	message input
$\overline{u}\langle M\rangle.P$	message output

The null process 0 does nothing; $P \mid Q$ is the parallel composition of processes P and Q, used to represent participants of a protocol running in parallel; and replication $!P$ is the infinite composition $P \mid P \mid \ldots$, which is often used to capture an unbounded number of sessions. Name restriction $\nu\, n.P$ binds n inside P, the introduction of restricted names (or private names) is useful to capture both fresh random numbers (modelling nonces and keys, for example) and private channels. The conditional if $M = N$ then P else Q is standard, but we stress $M = N$ represents equality (modulo an equational theory) rather than strict syntactic identity. For convenience we abbreviate conditionals as if $M = N$ then P, when Q is the null process. Finally, communication is captured by message input and message output. The process $u(x).P$ awaits a message from channel u and then behaves as P with the received message bound to the variable x; that is, every free occurrence of x in P refers to the message received. The process $\overline{u}\langle M\rangle.P$ is ready to send M on channel u and then run P. In both of these cases we may omit P when it is 0. We sometimes write let $x = M$ in P in place of $P\{^{M}/_{x}\}$, that is, P with all free occurrences of x replaced by M. In such a substitution, we insist that no name n occurring in M becomes bound by a restriction $\nu\, n$ occurring in P; for example, let $x = c$ in $\nu\, c.\overline{c}\langle x\rangle$ is not allowed, but let $x = c$ in $\nu\, c'.\overline{c'}\langle x\rangle$ is permitted.

Bracketing must be used to avoid ambiguities in the way processes are written down. For example, the process $!\,P \mid Q$ might be interpreted as $(!P) \mid Q$, or as $!(P \mid Q)$. These processes are different. To avoid too much bracketing, we adopt conventions about the precedence of process operators. Unary operators $!$, $\nu\, n$, $u(x)$, and $\overline{u}\langle M\rangle$ bind more closely than binary operators; and the binary if-then-else operator binds more closely than the binary operator $|$. It follows that the expression $c(x).\text{if } x = M \text{ then } P \mid !Q \mid R$ means $(c(x).\text{if } x = M \text{ then } P) \mid (!Q) \mid R$. We remark that different conventions are used elsewhere, for example [25].

The expression $P \mid Q \mid R$ is also ambiguous, since it could mean either $(P \mid Q) \mid R$ or $P \mid (Q \mid R)$. However, we will later see that these processes are semantically identical, so we tolerate the ambiguity in the syntax. Another possible ambiguity arises because of the convention of omitting "else 0" in the if-then-else construct: it is not clear which "if" the "else" applies to in the expression:

$$\text{if } M = N \text{ then if } K = L \text{ then } Q \text{ else } R.$$

In absence of brackets indicating the contrary, we adopt the convention that the else branch belongs to the closest if and hence the statement should be interpreted as if $M = N$ then (if $K = L$ then Q else R).

Processes are extended with *active substitutions* to capture the knowledge exposed

to the adversarial environment:

$A, B, C ::=$	extended processes
P	plain process
$A \mid B$	parallel composition
$\nu\, n.A$	name restriction
$\nu\, x.A$	variable restriction
$\{^M/_x\}$	active substitution

The active substitution $\{^M/_x\}$ represents a process that has previously output M. The value M is now available to the environment by reference to the 'handle' x. The active substitution $\{^M/_x\}$ can replace the variable x for the term M in every process it comes into contact with. This behaviour can be controlled by restriction, and the process $\nu\, x.(\{^M/_x\} \mid P)$ corresponds exactly to let $x = M$ in P. This allows access to terms which the environment cannot construct; such a scenario may arise, for example, when the environment does not know all of the names occurring in a term. Arbitrarily large active substitutions can be obtained by parallel composition and we occasionally abbreviate $\{^{M_1}/_{x_1}\} \mid \ldots \mid \{^{M_l}/_{x_l}\}$ as $\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$ or $\{^{\tilde{M}}/_{\tilde{x}}\}$. We also use σ and τ to range over active substitutions. Extended processes must have at most one active substitution for each variable and there is exactly one when the variable is under restriction. In particular, this means that $\nu\, x$ can only occur if there is an active substitution $\{^M/_x\}$ in its scope. Finally, we write $\nu\, \tilde{u}$ for the (possibly empty) series of pairwise-distinct binders $\nu\, u_1.\nu\, u_2. \ldots .\nu\, u_l$.

Active substitutions are different from syntactic substitions. Firstly, active substitutions are processes, whereas syntactic substitions are not. In addition, their semantic behaviours differ. Syntactic substitutions can be applied directly to processes; for example, the application of a syntactic substitution σ to the free variables of an extended process A is written $A\sigma$. By contrast, active substitutions cannot be applied directly to processes (or terms). Active and syntactic substitutions are always assumed to be cycle-free: given substitution σ, the repeated application of σ to itself may only result in a bounded number of replacements. For example, the substitution $\{^{g(y)}/_x, ^n/_y\}$ is cycle free; but $\{^{g(y)}/_x, ^x/_y\}$ is not, since $\{^{g(y)}/_x, ^x/_y\}$ will result in the variable y referring to a term of infinite length.

The type system for terms is extended to processes. It enforces that M, N are of the same type in the conditional expression if $M = N$ then P else Q; message input $u(x)$ is defined only where u is of type $\mathsf{Channel}\langle\omega\rangle$ and x is of type ω; and similarly message output $\overline{u}\langle M\rangle$ requires u of type $\mathsf{Channel}\langle\omega\rangle$ and M of type ω. In addition, we assume that in active substitutions $\{^M/_x\}$, the term M and the variable x are of the same base type. This assumption was not explicitly stated in [8] but its necessity has been confirmed [9] and is essential for the validity of Theorem 1 [16]. Finally, we assume extended processes are well-typed.

The *scope* of names and variables are delimited by binders $u(x)$ and $\nu\, u$. The set of bound names $\mathrm{bn}(A)$ contains every name n which is under restriction $\nu\, n$ inside A. The set of bound variables $\mathrm{bv}(A)$ consists of all those variables x occurring in A that are bound by restriction $\nu\, x$ or input $u(x)$. We also define the set of free names and the set of free variables. The set of free names in A, denoted $\mathrm{fn}(A)$, consists of those names n occurring in A not in the scope of the binder $\nu\, n$. The set of free variables $\mathrm{fv}(A)$ contains the variables x occurring in A which are not in the scope of a restriction $\nu\, x$ or input

$u(x)$. Note that a name or variable can occur both free and bound in A. Thus, the sets $\text{fv}(A) \cap \text{bv}(A)$ and $\text{fn}(A) \cap \text{bn}(A)$ may be non-empty. We occasionally write $\text{fn}(M)$ and $\text{fv}(M)$, for the set of names, respectively variables, which appear in term M. The concept of bound and free values is similar to local and global scope in programming languages. An extended process is *closed* when every variable x is either bound or defined by an active substitution, (that is, the process contains $\{^M/_x\}$ for some term M). Example 1 demonstrates name and variable scope. Bound names and bound variables are subject to α-conversion (also called α-renaming); that is, they may be uniformly renamed without changing the meaning of the process, as demonstrated in Example 2.

A *frame*, denoted φ or ψ, is an extended process built from 0 and active substitutions $\{^M/_x\}$; which are composed by parallel composition and restriction. The *domain* $\text{dom}(\varphi)$ of a frame φ is the set of variables that φ exports; that is, the set of variables x for which φ contains an active substitution $\{^M/_x\}$ such that x is not under restriction. Every extended process A can be mapped to a frame $\varphi(A)$ by replacing every plain process in A with 0. The frame $\varphi(A)$ represents the static knowledge output by a process to its environment. The domain $\text{dom}(A)$ of A is the domain of $\varphi(A)$.

Example 1 (Name and variable scope) *Consider the closed process* $A \triangleq \nu\, x.((c(y).\ \overline{c}\langle \mathsf{aenc}(\mathsf{pair}(x,z),y)\rangle) \mid (\nu\, s.\{^s/_x\}) \mid \{^{h(x,s')}/_z\})$ *defined with respect to the signature* $\Sigma = \{\mathsf{aenc}, \mathsf{h}, \mathsf{pair}\}$ *where* $\mathsf{aenc}, \mathsf{h}, \mathsf{pair}$ *are all binary functions. The occurrences of the variable* x *are bound by the variable restriction* $\nu\, x$ *and those of* y *are bound by the input* $c(y)$*. The name* s *is bound by the name restriction* $\nu\, s$*. The remaining name* s' *and variable* z *occur free. To summarise, we have* $\text{fv}(A) = \{z\}$*,* $\text{bv}(A) = \{x, y\}$*,* $\text{fn}(A) = \{s'\}$ *and* $\text{bn}(A) = \{s\}$*. Now let* $A' \triangleq A \mid \{^s/_y\}$*, and observe that the occurrences of* s *and* y *in the active substitution* $\{^s/_y\}$ *are free. They have no relation to the bound values* s, y *which occur in* A*. To avoid confusion it is good practice to use distinct identifiers for free and bound, names and variables.*

Example 2 (α-conversion) *α-conversion means renaming a bound name or variable without changing the semantics. This is also done in logic (the statements* $\forall x.P(x)$ *and* $\forall y.P(y)$ *are equivalent), and in programming (the programs* `for i in I do P(i)` *and* `for j in I do P(j)` *are equivalent). In the applied pi calculus, the process* $\nu\, k.c(x).\overline{c}\langle \mathsf{senc}(k,x)\rangle$ *is equivalent to* $\nu\, s.c(y).\overline{c}\langle \mathsf{senc}(s,y)\rangle$*; we have* α*-renamed all occurrences of the name* k *to the name* s *and similarly the variable* x *has been renamed* y*. But,* $\nu\, k.c(x).\nu\, k.\overline{c}\langle \mathsf{senc}(k,x)\rangle$ *is not* α*-equivalent to* $\nu\, k.c(y).\nu\, s.\overline{c}\langle \mathsf{senc}(k,y)\rangle$*.*

2.1.1. Modelling the Handshake protocol

The Handshake protocol (Section 1.1) is defined with respect to the signature $\Sigma_H = \{\mathsf{true}, \mathsf{fst}, \mathsf{snd}, \mathsf{hash}, \mathsf{pk}, \mathsf{getmsg}, \mathsf{pair}, \mathsf{sdec}, \mathsf{senc}, \mathsf{adec}, \mathsf{aenc}, \mathsf{sign}, \mathsf{checksign}, \mathsf{mac}\}$ where true is a constant; $\mathsf{fst}, \mathsf{snd}, \mathsf{hash}, \mathsf{pk}, \mathsf{getmsg}$ are unary functions; and $\mathsf{pair}, \mathsf{sdec}, \mathsf{senc}, \mathsf{adec}, \mathsf{aenc}, \mathsf{sign}, \mathsf{checksign}, \mathsf{mac}$ are binary functions. The behaviour of these functions is captured by the smallest equational theory E_H satisfying the following equations over variables x, y:

$$
\begin{array}{ll}
\mathsf{fst}(\mathsf{pair}(x,y)) & = x \\
\mathsf{snd}(\mathsf{pair}(x,y)) & = y \\
\mathsf{sdec}(x,\mathsf{senc}(x,y)) & = y \\
\mathsf{adec}(x,\mathsf{aenc}(\mathsf{pk}(x),y)) & = y \\
\mathsf{getmsg}(\mathsf{sign}(x,y)) & = y \\
\mathsf{checksign}(\mathsf{pk}(x),\mathsf{sign}(x,y)) & = \mathsf{true}
\end{array}
$$

This theory allows us to model: pairing, both symmetric and asymmetric cryptography, digital signature schemes with message recovery, hash functions and message authentication codes. For example, in order to express that the application of the symmetric decryption function sdec to the term modelling a symmetric encryption $\mathsf{senc}(k,m)$ should return the plaintext m if the correct key k is supplied, we use the equation $\mathsf{sdec}(x,\mathsf{senc}(x,y)) = y$. The absence of any equational theory associated with the hash function ensures preimage resistance, second-preimage resistance and collision resistance properties of cryptographic hash functions (in fact, far stronger properties are ensured); and similar properties for the function mac. Observe that a tuple $M_1,\ldots,M_l$ of an arbitrary length l can be constructed by pairing $\mathsf{pair}(M_1,\mathsf{pair}(M_2,\mathsf{pair}(\ldots,\mathsf{pair}(M_{l-1},M_l)\ldots)))$ and an element can be extracted using the equations defined for fst, snd.

The Handshake protocol can now be captured in our calculus as the process P, defined as follows.

$$
\begin{array}{rl}
P & \triangleq \nu\, sk_{\mathcal{S}}.\nu\, sk_{\mathcal{C}}.\nu\, s. \\
& \quad \text{let } pk_S = \mathsf{pk}(sk_{\mathcal{S}}) \text{ in let } pk_C = \mathsf{pk}(sk_{\mathcal{C}}) \text{ in} \\
& \quad (\overline{c}\langle pk_S\rangle \mid \overline{c}\langle pk_C\rangle \mid !P_{\mathcal{S}} \mid !P_{\mathcal{C}})
\end{array}
$$

$$
\begin{array}{rl}
P_{\mathcal{S}} & \triangleq c(x\_pk).\nu\, k.\overline{c}\langle \mathsf{aenc}(x\_pk,\mathsf{sign}(sk_{\mathcal{S}},k))\rangle. \\
& \quad c(z).\text{if } \mathsf{fst}(\mathsf{sdec}(k,z)) = tag \text{ then } Q
\end{array}
$$

$$
\begin{array}{rl}
P_{\mathcal{C}} & \triangleq c(y).\text{let } y' = \mathsf{adec}(sk_{\mathcal{C}},y) \text{ in let } y\_k = \mathsf{getmsg}(y') \text{ in} \\
& \quad \text{if } \mathsf{checksign}(pk_S,y') = \mathsf{true} \text{ then} \\
& \quad \overline{c}\langle \mathsf{senc}(y\_k,\mathsf{pair}(tag,s))\rangle
\end{array}
$$

The process begins by constructing the private keys $sk_{\mathcal{C}}$, $sk_{\mathcal{S}}$ for principals $\mathcal{C}$, $\mathcal{S}$ respectively. The public key parts $\mathsf{pk}(sk_{\mathcal{C}})$, $\mathsf{pk}(sk_{\mathcal{S}})$ are then output on the public communication channel c, ensuring they are available to the adversary. (Observe that this is done using handles pk_C and pk_S for convenience.) The protocol then launches multiple copies of processes $P_{\mathcal{C}}$, $P_{\mathcal{S}}$ representing multiple sessions of the roles of $\mathcal{C}$ and $\mathcal{S}$. Note that syntactic scope does not represent the knowledge of a protocol's participants. For example, the server's private key $sk_{\mathcal{S}}$ is assumed not to be known by the client $\mathcal{C}$ (hence it does not occur in $P_{\mathcal{C}}$), even though $sk_{\mathcal{S}}$ is in the scope of $P_{\mathcal{C}}$.

We assume that $\mathcal{S}$ is willing to run the protocol with any other principal; the choice of her interlocutor will be made by the environment. This is captured by modelling the first input $c(x\_pk)$ to $P_{\mathcal{S}}$ as the interlocutor's public key. $\mathcal{C}$ on the other hand only wishes to share his secret s with $\mathcal{S}$, and $\mathcal{C}$ is assumed to know $\mathcal{S}$'s public key; accordingly, $\mathcal{S}$'s public key is hard-coded into the process $P_{\mathcal{C}}$. We additionally assume that each principal is willing to engage in an unbounded number of sessions and hence $P_{\mathcal{C}}$, $P_{\mathcal{S}}$ are under replication.

Figure 1. Properties of equality modulo the equational theory

Given an equational theory E the following properties are satisfied for all terms L, M, N, functions f of arity l, substitutions σ and names m, n:

1. $M = N \in E \Rightarrow M =_E N$
2. Equivalence relation:
   - Reflexivity: $M =_E M$
   - Symmetry: $M =_E N \Rightarrow N =_E M$
   - Transitivity: $L =_E M \wedge M =_E N \Rightarrow L =_E N$
3. Application of function symbols: $M_1 =_E N_1 \wedge \cdots \wedge M_l =_E N_l \Rightarrow f(M_1, \ldots, M_l) =_E f(N_1, \ldots, N_l)$
4. Substitution of terms for variables: $M =_E N \Rightarrow M\sigma =_E N\sigma$
5. Bijective renaming of names: $M =_E N \Rightarrow M\{^m/_n\} =_E N\{^m/_n\}$, where $m \notin (\mathrm{fn}(M) \cup \mathrm{fn}(N))$.

On request from her interlocutor, server $\mathcal{S}$ starts the protocol by selecting key k and outputting $\mathsf{aenc}(x\_pk, \mathsf{sign}(sk_{\mathcal{S}}, k))$; that is, her signature on the key k encrypted with her interlocutor's public key x_pk. Meanwhile $\mathcal{C}$ awaits the input of his interlocutor's signature on the key k encrypted using his public key. $\mathcal{C}$ decrypts the message and verifies the signature. Next, if $\mathcal{C}$ believes he is indeed talking to $\mathcal{S}$, he outputs his secret s encrypted with the symmetric key k. Note that he inserts a tag (modelled as a free name), so that the decryptor can verify that the decryption has worked correctly. Principal $\mathcal{S}$ inputs z and confirms the presence of the tag. Finally, principal $\mathcal{S}$ executes the process Q. The description of Q is independent of the protocol, but one would expect the recovery of the interlocutor's secret; that is, Q is defined by the process let $z\_s = \mathsf{snd}(\mathsf{sdec}(k, z))$ in Q' for some Q'.

The purpose of the protocol is to establish a session key to transport the secret represented by s. We abstract away from the details of what s is, and how many such secrets there are, by modelling it simply as a restricted name.

2.2. Operational semantics

The signature Σ is equipped with an equational theory E, that is, a set of equations of the form $M = N$, where terms M, N are defined over the signature Σ (sometimes written $M, N \in T_\Sigma$). This allows us to capture relationships between primitives defined in Σ. We define equality modulo the equational theory, written $=_E$, as the smallest equivalence relation on terms, that contains E and is closed under application of function symbols, substitution of terms for variables and bijective renaming of names. The properties of $=_E$ are summarised in Figure 1. We write $M =_E N$ when the equation $M = N$ is in the theory E and keep the signature implicit. When E is clear from the context we may abbreviate $M =_E N$ as $M = N$. The negation of $M =_E N$ is denoted $M \neq_E N$ (and similarly abbreviated $M \neq N$). For further discussion on equational theories see [8, §3] and [7].

Example 3 (Reasoning with equational theories) *Consider the signature Σ_H and the equational theory E_H. Show that each of the following equalities hold:*

1. $\mathsf{sdec}(k, \mathsf{senc}(k, L)) =_{E_H} L$
2. $\mathsf{pair}(M, N) =_{E_H} \mathsf{pair}(\mathsf{sdec}(k, \mathsf{senc}(k, \mathsf{fst}(\mathsf{pair}(M, N)))), N)$
3. $\mathsf{fst}(\mathsf{sdec}(\mathsf{snd}(\mathsf{pair}(K, L)), \mathsf{senc}(\mathsf{fst}(\mathsf{pair}(L, N)), \mathsf{pair}(M, K)))) =_{E_H} M$

Hint. Figure 1 may be helpful.

Contexts may be used to represent the adversarial environment in which a process is run; that environment provides the data that the process inputs, and consumes the data that it outputs. We define *context* $C[\_]$ to be an extended process with a hole. We obtain $C[A]$ as the result of filling $C[\_]$'s hole with the extended process A. An *evaluation context* is a context whose hole is not in the scope of a replication, a conditional, an input, or an output. A context $C[\_]$ closes A when $C[A]$ is closed.

2.2.1. Structural equivalence

Informally, two processes are structurally equivalent if they model the same thing, but the grammar permits different encodings. For example, to describe a pair of processes A, B running in parallel, the grammar forces us to put one on the left and one on the right, that is, we have to write either $A \mid B$, or $B \mid A$. These two processes are said to be structurally equivalent. Formally, *structural equivalence* ($\equiv$) is the smallest equivalence relation on extended processes that is closed by α-conversion of both bound names and bound variables, and closed under application of evaluation contexts such that:

$$
\begin{array}{lrl}
\textsc{Par-0} & A & \equiv A \mid 0 \\
\textsc{Par-A} & A \mid (B \mid C) & \equiv (A \mid B) \mid C \\
\textsc{Par-C} & A \mid B & \equiv B \mid A \\
\textsc{Repl} & !P & \equiv P \mid !P \\
 & & \\
\textsc{New-0} & \nu\, n.0 & \equiv 0 \\
\textsc{New-C} & \nu\, u.\nu\, w.A & \equiv \nu\, w.\nu\, u.A \\
\textsc{New-Par} & A \mid \nu\, u.B & \equiv \nu\, u.(A \mid B) \\
 & & \quad \text{where } u \notin \mathrm{fv}(A) \cup \mathrm{fn}(A) \\
 & & \\
\textsc{Alias} & \nu\, x.\{^M/_x\} & \equiv 0 \\
\textsc{Subst} & \{^M/_x\} \mid A & \equiv \{^M/_x\} \mid A\{^M/_x\} \\
\textsc{Rewrite} & \{^M/_x\} & \equiv \{^N/_x\} \\
 & & \quad \text{where } M =_E N
\end{array}
$$

The rules for parallel composition, replication and restriction are self-explanatory. ALIAS enables the introduction of an arbitrary active substitution with restricted scope. SUBST describes the application of an active substitution to a process that it comes into contact with. The rule helps demonstrate the distinction between syntactic and active substitutions with respect to processes. In the process $\{^M/_x\} \mid A$, the expression $\{^M/_x\}$ is a subprocess, whose meaning is that x is a handle for the environment to refer to M. In the process $A\{^M/_x\}$, the expression $\{^M/_x\}$ is a substitution meaning that free occurrences of x in A should be replaced by M (in such a way that no names or variables in M become bound). The final rule, REWRITE, allows terms that are equal modulo the equational theory to be swapped as desired.

Structural equivalence allows every closed extended process A to be rewritten as a substitution and a closed plain process with some restricted names: $A \equiv \nu\,\tilde{n}.(\{^{\tilde{M}}/_{\tilde{x}}\} \mid P)$ where $\mathrm{fv}(\tilde{M}) = \mathrm{fv}(P) = \emptyset$ and $\tilde{n} \subseteq \mathrm{fn}(\tilde{M})$. It follows immediately that every closed frame φ can be rewritten as a substitution with some restricted names: $\varphi \equiv \nu\,\tilde{n}.\{^{\tilde{M}}/_{\tilde{x}}\}$ where $\mathrm{fv}(\tilde{M}) = \emptyset$ and $\tilde{n} \subseteq \mathrm{fn}(\tilde{M})$. We note that the domain of φ is $\tilde{x}$.

2.2.2. Internal reduction.

A process can be executed without contact with its environment, either because if-statements are evaluated and the then- or else-branch is taken, or because internal subprocesses communicate with each other. The execution of a process with respect to control flow and communication is captured by *internal reduction*. Formally, internal reduction ($\rightarrow$) is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that:

COMM	$\overline{c}\langle x\rangle.P \mid c(x).Q \rightarrow P \mid Q$
THEN	if $N = N$ then P else $Q \rightarrow P$
ELSE	if $L = M$ then P else $Q \rightarrow Q$ for ground terms L, M where $L \neq_E M$

Communication (COMM) is defined on variables, making it look rather restricted. However, this entails no loss of generality because ALIAS and SUBST can be used to allow communication of an arbitrary term M instead of a variable x. To see how this works, consider the process $\overline{c}\langle M\rangle.P \mid c(x).Q$. We can suppose that x is not in $\mathrm{fv}(P)$ (if it is, pick any $x' \notin \mathrm{fv}(P)$ and α-rename the bound variable x to x' in $c(x).Q$). Then $\overline{c}\langle M\rangle.P \mid c(x).Q \equiv \nu\,x.(\overline{c}\langle x\rangle.P \mid c(x).Q \mid \{^{M}/_{x}\})$. Since $\overline{c}\langle x\rangle.P \mid c(x).Q \rightarrow P \mid Q$ by COMM, we derive $P \mid Q\{^{M}/_{x}\}$ by application of an evaluation context. To see this step in more detail, consider $C[\_] = \nu\,x.(\_ \mid \{^{M}/_{x}\})$ and observe $\nu\,x.(\overline{c}\langle x\rangle.P \mid c(x).Q \mid \{^{M}/_{x}\}) = C[\overline{c}\langle x\rangle.P \mid c(x).Q] \rightarrow C[P \mid Q]$ which is structurally equivalent to $P \mid Q\{^{M}/_{x}\}$. Since $\rightarrow$ is closed under structural equivalence, we have $\overline{c}\langle M\rangle.P \mid c(x).Q \rightarrow P \mid Q\{^{M}/_{x}\}$.

Conditionals (THEN and ELSE) are dependent on the equational theory. Applications of THEN may require the use of the structural equivalence rule REWRITE to derive "$N = N$" from "$M = N$" where $M =_E N$. ELSE may require that active substitutions in the context be applied using ALIAS and SUBST to ensure L, M are ground.

2.2.3. Labelled reductions

The semantics in Section 2.2.2 allow us to reason about protocols with an adversary represented by a context. In order to prove that security properties hold for all adversaries, quantification over all contexts is typically required, which can be difficult in practice. The labelled operational semantics we now present aim to eliminate universal quantification of the context.

The labelled semantics defines a ternary relation written, $A \xrightarrow{\alpha} B$, where α is a *label* of the form $c(M)$, $\overline{c}\langle u\rangle$, or $\nu\,u.\overline{c}\langle u\rangle$ such that u is either a channel name or a variable of base type. The transition $A \xrightarrow{c(M)} B$ means that the process A performs an input of the term M from the environment on the channel c, and the resulting process is B. The situation for output is a bit more complicated, since there are several cases. If the item

is a free variable x or a free channel name d, then the label $\overline{c}\langle x\rangle$, respectively $\overline{c}\langle d\rangle$, is used. If the item being output is a restricted channel name d, then the label $\nu\, d.\overline{c}\langle d\rangle$ is used. Finally, if the item is a term M, then the label $\nu\, x.\overline{c}\langle x\rangle$ is used, after replacing the occurrence of the term M by x and wrapping the process in $\nu x.(\{^M/_x\} \mid \_)$. The operational semantics of §2.2.1 are extended to include the following rules:

$$\textsc{In} \qquad c(x).P \xrightarrow{c(M)} P\{^M/_x\}$$

$$\textsc{Out-Atom} \qquad \overline{c}\langle u\rangle.P \xrightarrow{\overline{c}\langle u\rangle} P$$

$$\textsc{Open-Atom} \qquad \frac{A \xrightarrow{\overline{c}\langle u\rangle} A' \qquad u \neq c}{\nu\, u.A \xrightarrow{\nu\, u.\overline{c}\langle u\rangle} A'}$$

$$\textsc{Scope} \qquad \frac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu\, u.A \xrightarrow{\alpha} \nu\, u.A'}$$

$$\textsc{Par} \qquad \frac{A \xrightarrow{\alpha} A' \qquad \mathrm{bv}(\alpha) \cap \mathrm{fv}(B) = \mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\textsc{Struct} \qquad \frac{A \equiv B \qquad B \xrightarrow{\alpha} B' \qquad B' \equiv A'}{A \xrightarrow{\alpha} A'}$$

The rule OPEN-ATOM is used in two ways: 1) to output a restricted channel; and 2) to output a term. The first instance is straight forward and one may prove that $\nu d.\overline{c}\langle d\rangle.P \xrightarrow{\nu d.\overline{c}\langle d\rangle} P$, provided $c \neq d$. The latter is a little more complicated since it requires the use of structural equivalence rules. Observe that $\overline{c}\langle M\rangle.P \xrightarrow{\nu x.\overline{c}\langle x\rangle} P \mid \{^M/_x\}$, where $x \notin \mathrm{fv}(P)$, by first writing $\overline{c}\langle M\rangle.P$ as $\nu x.(\overline{c}\langle x\rangle.P \mid \{^M/_x\})$. The full derivation is given below:

$$\dfrac{\overline{c}\langle M\rangle.P \equiv \nu\, x.(\overline{c}\langle x\rangle.P \mid \{^M/_x\}) \xrightarrow{\nu\, x.\overline{c}\langle x\rangle} P \mid \{^M/_x\} \equiv P \mid \{^M/_x\}}{\overline{c}\langle M\rangle.P \xrightarrow{\nu\, x.\overline{c}\langle x\rangle} P \mid \{^M/_x\}}\ \textsc{Struct}$$

$$\dfrac{\dfrac{\dfrac{}{\overline{c}\langle x\rangle.P \xrightarrow{\overline{c}\langle x\rangle} P}\ \textsc{Out-Atom}}{\overline{c}\langle x\rangle.P \mid \{^M/_x\} \xrightarrow{\overline{c}\langle x\rangle} P \mid \{^M/_x\}}\ \textsc{Par}}{\nu\, x.(\overline{c}\langle x\rangle.P \mid \{^M/_x\}) \xrightarrow{\nu\, x.\overline{c}\langle x\rangle} P \mid \{^M/_x\}}\ \textsc{Open-Atom}$$

Note that the fact $x \notin \mathrm{fv}(P)$ is needed for the first of the two structural equivalences in the occurrence of STRUCT.

Example 4 (Labelled semantics) *Consider the process $A \triangleq \overline{c}\langle m\rangle.\overline{c}\langle m\rangle$. Show the reductions $A \xrightarrow{\nu\, x.\overline{c}\langle x\rangle} \xrightarrow{\nu\, y.\overline{c}\langle y\rangle} \{^m/_x\} \mid \{^m/_y\}$. Also show the reductions $A \xrightarrow{\nu\, x.\overline{c}\langle x\rangle} \xrightarrow{\overline{c}\langle x\rangle} \{^m/_x\}$. (Hint. The last one involves an application of* SUBST.*)*

2.2.4. Names and the environment

The reduction $\nu\, \tilde{n}.\overline{c}\langle M\rangle.P \xrightarrow{\nu x.\overline{c}\langle x\rangle} \nu\, \tilde{n}.(P \mid \{^M/_x\})$ represents an important idiom of the applied pi calculus. It is the way that an arbitrary term M is output to the environment. This illustrates the way that the environment's use of names is controlled.

If s is a restricted name in a process then the environment cannot use that s to construct a term. Moreover, if the environment uses the name s, then it is not the same as the one in the process. Consider for example the process

$$A \triangleq \nu\, s.(c(x).\text{if } x = s \text{ then } \overline{c}\langle i\_got\_s\rangle)$$

This process can never output i_got_s, because no term input as x can be equal to the 'new' s created by the process. More precisely, there is no sequence of reductions

$$A \rightarrow^* \xrightarrow{\alpha} \rightarrow^* \cdots \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B \mid \{^{i\_got\_s}/_y\}$$

for some process B and variable y.

Now suppose the process A' creates a new s and outputs some term containing s:

$$A' \triangleq \nu\, s.(\overline{c}\langle \mathsf{hash}(s)\rangle.B')$$

We have $A' \xrightarrow{\nu\, x.\overline{c}\langle x\rangle} \nu\, s.(B' \mid \{^{\mathsf{hash}(s)}/_x\})$. This process exposes $\mathsf{hash}(s)$ to the environment, by reference to the handle x. Although the environment still does not have s, it can use $\mathsf{hash}(s)$ in an expression, simply by using x in its place. For example, it can use x to construct input in B'.

Now consider the process A'' which creates a new s, outputs the encryption of s by a (free) key k, and then accepts an input and tests if the input is s.

$$A'' \triangleq \nu\, s.(\overline{c}\langle \mathsf{senc}(k, s)\rangle.c(x).\text{if } x = s \text{ then } \overline{c}\langle i\_got\_s\rangle)$$

This test can succeed; the process can output i_got_s, as shown by the following execution:

$$\begin{aligned}
A'' &\xrightarrow{\nu\, y.\overline{c}\langle y\rangle} && \nu\, s.(c(x).\text{if } x = s \text{ then } \overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,\, s)}/_y\}) \\
&\xrightarrow{c(\mathsf{sdec}(k,y))} && \nu\, s.(\text{if } \mathsf{sdec}(k, y) = s \text{ then } \overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,\, s)}/_y\}) \\
&\equiv && \nu\, s.(\text{if } \mathsf{sdec}(k, \mathsf{senc}(k, s)) = s \text{ then } \overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,\, s)}/_y\}) \\
&\equiv && \nu\, s.(\text{if } s = s \text{ then } \overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,\, s)}/_y\}) \\
&\rightarrow && \nu\, s.(\overline{c}\langle i_got_s\rangle \mid \{^{\mathsf{senc}(k,\, s)}/_y\}) \\
&\xrightarrow{\nu\, z.\overline{c}\langle z\rangle} && \nu\, s.(\{^{\mathsf{senc}(k,\, s)}/_y\} \mid \{^{i_got_s}/_z\}) \\
&\equiv && \nu\, s.(\{^{\mathsf{senc}(k,\, s)}/_y\}) \mid \{^{i_got_s}/_z\}
\end{aligned}$$

The first of the equivalences $\equiv$ holds by the rule SUBST; the second one is by REWRITE, using the equation $\mathsf{sdec}(k, \mathsf{senc}(k, s)) = s$. Intuitively, the environment has taken the encrypted output and decrypted it (since the key k is a name known to the environment). It can then input the result to the process, which sees that it is indeed equal to s.

It is instructive to return to the process $A \triangleq \nu\, s.(c(x).\text{if } x = s \text{ then } \overline{c}\langle i\_got\_s\rangle)$ above, and consider what happens if the environment gives it the name s as input. As stated earlier, intuitively this s from the environment is considered different from the s that the process constructs. Technically, this is handled as follows. If we try to perform a transition $A \xrightarrow{c(s)} B$, we find that we cannot quite use the expected combination of the rules IN and SCOPE, because IN asks us to perform the substitution

$$\nu\, s.(\text{if } x = s \text{ then } \overline{c}\langle i\_got\_s\rangle\{^s/_x\})$$

which would insert the s from the substitution into the scope of the νs. As previously mentioned, such substitutions are not allowed. We can resolve the situation by α-renaming the bound s first:

$$A \equiv \nu\, s'.(c(x).\text{if } x = s' \text{ then } \overline{c}\langle i\_got\_s\rangle).$$

Recall that if $M = N$ then P is an abbreviation of if $M = N$ then P else 0 and hence we have the reductions:

$$\nu\, s'.(c(x).\text{if } x = s' \text{ then } \overline{c}\langle i\_got\_s\rangle) \xrightarrow{c(s)} \text{if } s = s' \text{ then } \overline{c}\langle i\_got\_s\rangle \to 0.$$

3. Secrecy and correspondence properties

This section presents formal definitions of secrecy and correspondence in the presence of an adversary who has full control of the network. The attacker can therefore eavesdrop, replay, inject and block messages. Formally the attacker is captured as an arbitrary process and is sometimes called the Dolev-Yao adversary [36].

3.1. Secrecy

Intuitively, a protocol preserves the secrecy of some term M if an adversary cannot obtain M by constructing it from the outputs of the protocol [3]. We formalise the adversary as a process running in parallel with the protocol, that after constructing M outputs it on a public channel. The adversary process does not have any of the protocol's secrets.

A term may refer to names inside a process; to analyse secrecy of the term, it is important that these names appear unambiguously in the process. Otherwise it is not clear which name the term refers to. For example, if a name in the term is both bound and free in the process, then it is not clear whether the term refers to the free instance, or the bound instance. We formalise that by requiring the process to be "name distinct" for the names mentioned in the term.

Definition 1 (name-distinct for $\tilde{m}$) *A closed plain process P is* name-distinct *for a set of names $\tilde{m}$, if $\tilde{m} \cap \text{fn}(P) \cap \text{bn}(P) = \emptyset$ and for each name $n \in \tilde{m} \cap \text{bn}(P)$ there is exactly one name restriction $\nu\, n$ occurring in P, and the restriction is not under replication "!".*

Definition 2 (Can output) *A plain process P* can output the term M *if there exists an evaluation context $C[\_]$, a channel name $c \notin \text{bn}(C)$ and process R such that the reduction $P \to^* C[\overline{c}\langle M\rangle.R]$ holds with no alpha-renaming of the names in* $\text{fn}(M)$.

In Definition 2, the process $C[\overline{c}\langle M\rangle.R]$ is capable of outputting the term M on the free channel c in one further step. Note that the definition forbids renaming names in M during the reduction $P \to^* C[\overline{c}\langle M\rangle.R]$, because we must not change which names in P are referred to by M. Thus, the process $\nu b.\overline{c}\langle b\rangle$ cannot output s; if we allowed renaming, we could write the process as $\nu s.\overline{c}\langle s\rangle$ and it would be able to output s.

Definition 3 (Reachability-based secrecy) *Let M be a term, and P be a closed plain process that is name-distinct for* $\mathrm{fn}(M)$*. Then P preserves the* reachability-based secrecy *of M if there is no plain process I such that* $(\mathrm{fn}(I) \cup \mathrm{bn}(I)) \cap \mathrm{bn}(P) = \emptyset$ *and $P \mid I$ can output M.*

In the definition above, I is the adversary (or *intruder*) process. Typically, it is built in order to receive the outputs from P, and then possibly to construct from them the secret term M, and output it. If there is no such intruder, then P keeps M secret. This definition is based on the one in [3], but extends it to cope with bound names.

Example 5 (Reasoning with secrecy) *Consider the process P corresponding to the Handshake protocol (Section 2.1.1) and the equational theory E_H. We show that P does not preserve the secrecy of s, by presenting an adversarial process*

$$\begin{aligned} I \triangleq\ & c(y\_pk).\overline{c}\langle \mathsf{pk}(sk_M)\rangle.c(x). \\ & \overline{c}\langle \mathsf{aenc}(y\_pk, \mathsf{adec}(sk_M, x))\rangle.c(z). \\ & \overline{c}\langle \mathsf{snd}(\mathsf{sdec}(\mathsf{getmsg}(\mathsf{adec}(sk_M, x)), z))\rangle \end{aligned}$$

and demonstrating that $P \mid I$ can evolve to a process that can output s on a public channel. To aid readability, we apply all the substitutions denoted by 'let' occurring in P, and we use the context

$$C[\_] \triangleq \nu\, sk_{\mathcal{S}}.\nu\, sk_{\mathcal{C}}.\nu\, s.\,(\_ \mid !P_{\mathcal{S}} \mid !P_{\mathcal{C}})$$

and write $P \equiv C[\overline{c}\langle \mathsf{pk}(sk_{\mathcal{S}})\rangle \mid \overline{c}\langle \mathsf{pk}(sk_{\mathcal{C}})\rangle \mid P_{\mathcal{S}} \mid P_{\mathcal{C}}]$. Since $sk_{\mathcal{S}}, sk_{\mathcal{C}}, s \notin \mathrm{fn}(I) \cup \mathrm{fv}(I)$, we have

$$\begin{aligned} P \mid I \equiv C[\ & \overline{c}\langle \mathsf{pk}(sk_{\mathcal{S}})\rangle \mid \overline{c}\langle \mathsf{pk}(sk_{\mathcal{C}})\rangle \\ & \mid c(x\_pk).\nu\, k.\overline{c}\langle \mathsf{aenc}(x\_pk, \mathsf{sign}(sk_{\mathcal{S}}, k))\rangle. \\ & \ \ c(z).\mathsf{if}\ \mathsf{fst}(\mathsf{sdec}(k, z)) = tag\ \mathsf{then}\ Q \\ & \mid c(y).\mathsf{if}\ \mathsf{checksign}(\mathsf{pk}(sk_{\mathcal{S}}), \mathsf{adec}(sk_{\mathcal{C}}, y)) = \mathsf{true}\ \mathsf{then} \\ & \ \ \overline{c}\langle \mathsf{senc}(\mathsf{getmsg}(\mathsf{adec}(sk_{\mathcal{C}}, y)), \mathsf{pair}(tag, s))\rangle \\ & \mid I\ ] \end{aligned}$$

Intuitively, the following sequence takes place:

1. *$\mathcal{C}$'s public key $\mathsf{pk}(sk_{\mathcal{C}})$ is published and hence made available to the adversary, who inputs it as y_{pk}.*
2. *The adversary provides her public key $\mathsf{pk}(sk_M)$ as $\mathcal{S}$'s interlocutor.*
3. *$\mathcal{S}$ outputs her signature on k, encrypted for the adversary, and the adversary inputs it as x. (Observe that closure under structural equivalence is used to move the name restriction $\nu\, k$ outside of the context; more precisely, the rules* NEW-C, NEW-PAR *are applied.)*
4. *The adversary gives to $\mathcal{C}$ the value k signed by $\mathcal{S}$, this time encrypted for $\mathcal{C}$.*
5. *The process is further rewritten using structural equivalence (essentially several occurrences of the rule* REWRITE *with respect to E_H). (Note that this rewriting could have been included in the previous step because internal reduction is closed under structural equivalence, but we explicitly present it for clarity.)*

6. *The conditional is evaluated and $\mathcal{C}$ sends the value* $\mathsf{senc}(k, \mathsf{pair}(tag, s))$ *to the adversary.*
7. *Observe that* $\overline{c}\langle \mathsf{snd}(\mathsf{sdec}(k, \mathsf{senc}(k, \mathsf{pair}(tag, s)))) \rangle \equiv \overline{c}\langle s \rangle$ *and hence the adversary can obtain* s *and publish the value on a public channel.*

The execution path which results in the adversary being able to output s *on a public channel is described in a detailed figure in the extended version of this chapter [44].*

3.2. Correspondence properties

Correspondence properties are used to capture relationships between events that can be expressed in the form *"if an event* e *has been executed then event* e' *has been previously executed."* Moreover, these events may contain arguments, which allow relationships between the arguments of events to be expressed. To reason with correspondence properties we annotate processes with *events*, marking important stages reached by the protocol which do not otherwise affect behaviour. Events are analogous to breakpoints used in software development. In this chapter, we only consider basic correspondence properties, which are sufficient to model authentication. More elaborate formalisms can be found in [5,21]. Events are message outputs $\overline{f}\langle M \rangle$ where f is an *event channel* (a name in a particular set, disjoint from the set of ordinary channel names). In labelled transitions, output labels for events use *event variables* e. Those event variables are not allowed to appear in input labels $u(M)$, so the adversary cannot use them. (This condition is important, since events are added just for checking correspondence properties; in particular an event $\overline{f}\langle M \rangle$ does not reveal M to the adversary.) Hence, the execution of the process P after inserting events $\overline{f}\langle M \rangle$ is the execution of P without events, plus the recording of events using labels $\nu\, e.\overline{f}\langle e \rangle$ and active substitutions $\{^{M}/_{e}\}$.

Definition 4 (Correspondence property) *A* correspondence property *is a formula of the form:* $\overline{f}\langle M \rangle \rightsquigarrow \overline{g}\langle N \rangle$.

The property asserts that if the event f has been executed in a trace with parameters M, then the event g must have been previously executed with parameters N.

Authentication can be captured as a correspondence property. Recall that in addition to the secrecy property mentioned for the Handshake protocol in Section 1, there were also authentication properties. The protocol is intended to ensure that if $\mathcal{S}$ thinks she executes the protocol with $\mathcal{C}$, then she really does so, and vice versa. When we say 'she thinks' that she executes it with $\mathcal{C}$, we mean that the data she receives indicates that fact. Accordingly we annotate the Handshake protocol with events. To capture equality tests within events we include the binary function eq in the signature Σ_H and extend the equational theory E_H with the equation $\mathsf{eq}(x, x) = \mathsf{true}$.

Example 6 (Annotated Handshake protocol) *The annotated Handshake protocol is presented below*

$$
\begin{aligned}
P \;&\triangleq \nu\, sk_{\mathcal{S}}.\nu\, sk_{\mathcal{C}}.\nu\, s.\\
&\quad \text{let } pk_S = \mathsf{pk}(sk_{\mathcal{S}}) \text{ in let } pk_C = \mathsf{pk}(sk_{\mathcal{C}}) \text{ in}\\
&\quad (\overline{c}\langle pk_S\rangle \mid \overline{c}\langle pk_C\rangle \mid !P_{\mathcal{S}} \mid !P_{\mathcal{C}})
\end{aligned}
$$

$$
\begin{aligned}
P_{\mathcal{S}} \;&\triangleq c(x\_pk).\nu\, k.\overline{startedS}\langle\mathsf{pair}(x\_pk, k)\rangle\\
&\quad \overline{c}\langle\mathsf{aenc}(x\_pk, \mathsf{sign}(sk_{\mathcal{S}}, k))\rangle.\\
&\quad c(z).\text{if } \mathsf{fst}(\mathsf{sdec}(k, z)) = tag \text{ then}\\
&\quad \overline{completedS}\langle\mathsf{pair}(k, \mathsf{eq}(x\_pk, pk_C))\rangle.Q
\end{aligned}
$$

$$
\begin{aligned}
P_{\mathcal{C}} \;&\triangleq c(y).\text{let } y' = \mathsf{adec}(sk_{\mathcal{C}}, y) \text{ in let } y\_k = \mathsf{getmsg}(y') \text{ in}\\
&\quad \overline{startedC}\langle y\_k\rangle\\
&\quad \text{if } \mathsf{checksign}(pk_S, y') = \mathsf{true} \text{ then}\\
&\quad \overline{c}\langle\mathsf{senc}(y\_k, \mathsf{pair}(tag, s))\rangle\\
&\quad \overline{completedC}\langle\mathsf{pair}(pk_C, y\_k)\rangle
\end{aligned}
$$

where the four events are interpreted as follows:

- $\overline{startedS}\langle\mathsf{pair}(x\_pk, k)\rangle$ *means that* $\mathcal{S}$ *considers she has started the protocol with an interlocutor whose public key is* x_pk*, and she has proposed* k *as the session key.*
- $\overline{startedC}\langle y\_k\rangle$ *means that* $\mathcal{C}$ *considers he has started the protocol with the session key* y_k*.*
- $\overline{completedS}\langle\mathsf{pair}(k, t)\rangle$ *means that* $\mathcal{S}$ *believes she has completed the protocol with session key* k *and if* $t =_{E_H} \mathsf{true}$*, then the protocol was completed with* $\mathcal{C}$*.*
- $\overline{completedC}\langle\mathsf{pair}(pk_{\mathcal{C}}, y\_k)\rangle$ *means that* $\mathcal{C}$ *considers he has successfully completed the protocol with* $\mathcal{S}$ *using session key* y_k*, where* $pk_{\mathcal{C}}$ *is his public key.*

Correspondence properties can now be defined to allow the analysis of authentication. Recall that the client $\mathcal{C}$ *is only willing to share her secret with the server* $\mathcal{S}$*. We formalise the authentication of* $\mathcal{C}$ *using the correspondence property*

$$\overline{completedC}\langle\mathsf{pair}(x, y)\rangle \rightsquigarrow \overline{startedS}\langle\mathsf{pair}(x, y)\rangle.$$

In comparison, $\mathcal{S}$ *is willing to run the protocol with any other principal. The correspondence property says that if she believes* $\mathcal{C}$ *was her interlocutor, then* $\mathcal{C}$ *must have completed the protocol with the suggested key. This is formalised as:*

$$\overline{completedS}\langle\mathsf{pair}(y, \mathsf{true})\rangle \rightsquigarrow \overline{startedC}\langle y\rangle$$

The subtle differences between the two correspondence properties is due to the differing authentication properties expected by participants $\mathcal{S}$ *and* $\mathcal{C}$*.*

Formally we define the validity of a correspondence property in Definition 5. Intuitively, it ensures that if the event f is executed, then the event g must have been previously executed. Moreover, the parametrisation of the events must satisfy any relationships defined by M, N; that is, the variables $\mathrm{fv}(M) \cap \mathrm{fv}(N)$ must be parametrised in the same way.

Definition 5 (Validity of correspondence property) *Let E be an equational theory, and A_0 an extended process. We say that A_0* satisfies the correspondence property *$\overline{f}\langle M\rangle \rightsquigarrow \overline{g}\langle N\rangle$ if for all execution paths*

$$A_0 \rightarrow^* \xrightarrow{\alpha_1} \rightarrow^* A_1 \rightarrow^* \xrightarrow{\alpha_2} \rightarrow^* \cdots \rightarrow^* \xrightarrow{\alpha_n} \rightarrow^* A_n,$$

and all index $i \in \mathbb{N}$, substitution σ and variable e such that $\alpha_i = \nu\, e.\overline{f}\langle e\rangle$ and $e\varphi(A_i) =_E M\sigma$, there exists $j \in \mathbb{N}$ and e' such that $\alpha_j = \nu\, e'.\overline{g}\langle e'\rangle$, $e'\varphi(A_j) =_E N\sigma$ and $j < i$.

Example 7 (Reasoning with correspondence) *Consider the annotated Handshake protocol (Example 6). The first correspondence property, namely $\overline{completedC}\langle \mathsf{pair}(x, y)\rangle \rightsquigarrow \overline{startedS}\langle \mathsf{pair}(x, y)\rangle$ is not valid. This can be observed by constructing an execution path in which the events $completedC$ and $startedS$ both occur, but with different arguments. To aid readability, we apply some of the substitutions denoted by 'let' occurring in P, and we reuse the context $C[\_] \triangleq \nu\, sk_{\mathcal{S}}.\nu\, sk_{\mathcal{C}}.\nu\, s.\,(\_ \mid !P_{\mathcal{S}} \mid !P_{\mathcal{C}})$ from Example 5. The execution path is shown in Figures 2 & 3. Intuitively, the following sequence of actions takes place (the item numbers correspond to the transitions in the figures):*

1. *$\mathcal{C}$'s public key $\mathsf{pk}(sk_{\mathcal{C}})$ is output, using the handle y_pk. This public key is now available to the environment.*
2. *The environment provides the public key $\mathsf{pk}(sk_M)$ as $\mathcal{S}$'s interlocutor.*
3. *The event $\overline{startedS}\langle \mathsf{pair}(x\_pk, k)\rangle$ is executed with the environment's public key $\mathsf{pk}(sk_M)$ assigned to parameter x_pk.*
4. *$\mathcal{S}$ outputs as x the value k signed by $\mathcal{S}$ and encrypted for the environment.*
5. *The environment gives to $\mathcal{C}$ the value k signed by $\mathcal{S}$, this time encrypted for $\mathcal{C}$.*
6. *The process is rewritten using structural equivalence (essentially several occurrences of the rules* SUBST *and* REWRITE*).*
7. *The event $\overline{startedC}\langle y\_k\rangle$ is executed with k as parameter y_k.*
8. *The conditional is trivially evaluated and $\mathcal{C}$ outputs the value $\mathsf{senc}(k, \mathsf{pair}(tag, s))$ as z.*
9. *Finally, the event $\overline{completedC}\langle \mathsf{pair}(pk_{\mathcal{C}}, y\_k)\rangle$ is executed with respect to the value k assigned to parameter y_k.*

Thus, the conditions of Definition 5 are violated; we have $\overline{completedC}\langle e_3\rangle$, and although we do have a previous $\overline{startedS}\langle e_1\rangle$, it does not have the same arguments as can be observed from the frame.

3.2.1. Injective correspondence properties

The definition of correspondence we have just discussed is insufficient to capture injective relationships between events; making it unsuitable for certain authentication properties. For example, consider a financial transaction in which a server requests payment from a client; the server should only complete a transaction, when that transaction was started by the client. (If this were not the case, the client could be charged for several transactions, even if the client only started one.) The situation is similar for access control and other scenarios. Further discussion can be found in [41].

Figure 2. Execution of Handshake protocol, part I (labelled semantics)

$$
\begin{array}{lll}
1. & P \xrightarrow{\nu\, y\_pk.\overline{c}\langle y\_pk\rangle} & C[\overline{c}\langle \mathsf{pk}(sk_{\mathcal{S}})\rangle \mid \{^{\mathsf{pk}(sk_{\mathcal{C}})}/_{y\_pk}\} \\
 & & \mid c(x\_pk).\nu\, k.\overline{startedS}\langle \mathsf{pair}(x\_pk, k)\rangle. \\
 & & \quad \overline{c}\langle \mathsf{aenc}(x\_pk, \mathsf{sign}(sk_{\mathcal{S}}, k))\rangle. \\
 & & \quad c(z).\mathsf{if}\ \mathsf{fst}(\mathsf{sdec}(k, z)) = tag\ \mathsf{then} \\
 & & \quad \overline{completedS}\langle \mathsf{pair}(k, \mathsf{eq}(x\_pk, \mathsf{pk}(sk_{\mathcal{C}})))\rangle.Q \\
 & & \mid c(y). \\
 & & \quad \mathsf{let}\ y' = \mathsf{adec}(sk_{\mathcal{C}}, y)\ \mathsf{in} \\
 & & \quad \mathsf{let}\ y\_k = \mathsf{getmsg}(y')\ \mathsf{in} \\
 & & \quad \overline{startedC}\langle y\_k\rangle. \\
 & & \quad \mathsf{if}\ \mathsf{checksign}(\mathsf{pk}(sk_{\mathcal{S}}), y') = \mathsf{true}\ \mathsf{then} \\
 & & \quad \overline{c}\langle \mathsf{senc}(y\_k, \mathsf{pair}(tag, s))\rangle \\
 & & \quad \overline{completedC}\langle \mathsf{pair}(\mathsf{pk}(sk_{\mathcal{C}}), y\_k)\rangle] \\
2. & \xrightarrow{c(\mathsf{pk}(sk_M))} & \\
3. & \xrightarrow{\nu\, e_1.\overline{startedS}\langle e_1\rangle} & \\
4. & \xrightarrow{\nu\, x.\overline{c}\langle x\rangle} & \nu\, k.C[\overline{c}\langle \mathsf{pk}(sk_{\mathcal{S}})\rangle \mid \{^{\mathsf{pk}(sk_{\mathcal{C}})}/_{y\_pk}\} \\
 & & \mid \{^{\mathsf{pair}(\mathsf{pk}(sk_M), k)}/_{e_1}\} \\
 & & \mid \{^{\mathsf{aenc}(\mathsf{pk}(sk_M), \mathsf{sign}(sk_{\mathcal{S}}, k))}/_{x}\} \\
 & & \mid c(z).\mathsf{if}\ \mathsf{fst}(\mathsf{sdec}(k, z)) = tag\ \mathsf{then} \\
 & & \quad \overline{completedS}\langle \mathsf{pair}(k, \mathsf{eq}(\mathsf{pk}(sk_M), \mathsf{pk}(sk_{\mathcal{C}})))\rangle.Q \\
 & & \mid c(y). \\
 & & \quad \mathsf{let}\ y' = \mathsf{adec}(sk_{\mathcal{C}}, y)\ \mathsf{in} \\
 & & \quad \mathsf{let}\ y\_k = \mathsf{getmsg}(y')\ \mathsf{in} \\
 & & \quad \overline{startedC}\langle y\_k\rangle. \\
 & & \quad \mathsf{if}\ \mathsf{checksign}(\mathsf{pk}(sk_{\mathcal{S}}), y') = \mathsf{true}\ \mathsf{then} \\
 & & \quad \overline{c}\langle \mathsf{senc}(y\_k, \mathsf{pair}(tag, s))\rangle \\
 & & \quad \overline{completedC}\langle \mathsf{pair}(\mathsf{pk}(sk_{\mathcal{C}}), y\_k)\rangle]
\end{array}
$$

Example 8 *Consider the process*

$$\overline{start}\langle n\rangle.\overline{complete}\langle n\rangle.\overline{complete}\langle n\rangle$$

which satisfies the correspondence $\overline{complete}\langle x\rangle \rightsquigarrow \overline{start}\langle x\rangle$*, but permits two occurrences of the event* $\overline{complete}\langle n\rangle$ *to be matched by a single event* $\overline{start}\langle n\rangle$.

Injective correspondence properties are denoted by the presence of the keyword inj.

Definition 6 (Injective correspondence property) *An* injective correspondence property *is a formula of the form:* $\overline{f}\langle M\rangle \rightsquigarrow \mathsf{inj}\ \overline{g}\langle N\rangle$.

Informally, it asserts that if a process executes event f, then there is a distinct earlier occurrence of the event g and any relationship between the event parameters is satisfied. It follows immediately that the number of occurrences of the label $\nu\, e.\overline{g}\langle e\rangle$ is greater than, or equal to, the number of occurrences of $\nu\, e'.\overline{f}\langle e'\rangle$ for some event variables e, e'.

Figure 3. Execution of Handshake protocol, part II (labelled semantics)

$$
\begin{array}{lll}
5. & \xrightarrow{c(\mathsf{aenc}(y\_pk,\mathsf{adec}(sk_M,x)))} & \nu\, k.C[\overline{c}\langle \mathsf{pk}(sk_{\mathcal{S}})\rangle \mid \{^{\mathsf{pk}(sk_{\mathcal{C}})}/_{y\_pk}\} \\
 & & \mid \{^{\mathsf{pair}(\mathsf{pk}(sk_M),k)}/_{e_1}\} \\
 & & \mid \{^{\mathsf{aenc}(\mathsf{pk}(sk_M),\mathsf{sign}(sk_{\mathcal{S}},k))}/_{x}\} \\
 & & \mid c(z).\text{if } \mathsf{fst}(\mathsf{sdec}(k,z)) = tag \text{ then} \\
 & & \overline{completedS}\langle \mathsf{pair}(k,\mathsf{eq}(\mathsf{pk}(sk_M),\mathsf{pk}(sk_{\mathcal{C}})))\rangle.Q \\
 & & \mid \text{let } y' = \mathsf{adec}(sk_{\mathcal{C}},\mathsf{aenc}(y\_pk,\mathsf{adec}(sk_M,x))) \text{ in} \\
 & & \text{let } y\_k = \mathsf{getmsg}(y') \text{ in} \\
 & & \overline{startedC}\langle y\_k\rangle. \\
 & & \text{if } \mathsf{checksign}(\mathsf{pk}(sk_{\mathcal{S}}),y') = \mathsf{true} \text{ then} \\
 & & \overline{c}\langle \mathsf{senc}(y\_k,\mathsf{pair}(tag,s))\rangle \\
 & & \overline{completedC}\langle \mathsf{pair}(\mathsf{pk}(sk_{\mathcal{C}}),y\_k)\rangle] \\
6. & \equiv & \nu\, k.C[\overline{c}\langle \mathsf{pk}(sk_{\mathcal{S}})\rangle \mid \{^{\mathsf{pk}(sk_{\mathcal{C}})}/_{y\_pk}\} \\
 & & \mid \{^{\mathsf{pair}(\mathsf{pk}(sk_M),k)}/_{e_1}\} \\
 & & \mid \{^{\mathsf{aenc}(\mathsf{pk}(sk_M),\mathsf{sign}(sk_{\mathcal{S}},k))}/_{x}\} \\
 & & \mid c(z).\text{if } \mathsf{fst}(\mathsf{sdec}(k,z)) = tag \text{ then} \\
 & & \overline{completedS}\langle \mathsf{pair}(k,\mathsf{eq}(\mathsf{pk}(sk_M),\mathsf{pk}(sk_{\mathcal{C}})))\rangle.Q \\
 & & \mid \overline{startedC}\langle k\rangle. \\
 & & \text{if } \mathsf{true} = \mathsf{true} \text{ then} \\
 & & \overline{c}\langle \mathsf{senc}(k,\mathsf{pair}(tag,s))\rangle \\
 & & \overline{completedC}\langle \mathsf{pair}(\mathsf{pk}(sk_{\mathcal{C}}),k)\rangle] \\
7. & \xrightarrow{\nu\, e_2.\overline{startedC}\langle e_2\rangle} & \\
8. & \rightarrow\xrightarrow{\nu\, z.\overline{c}\langle z\rangle} & \\
9. & \xrightarrow{\nu\, e_3.\overline{completedC}\langle e_3\rangle} & \nu\, k.C[\overline{c}\langle \mathsf{pk}(sk_{\mathcal{S}})\rangle \mid \{^{\mathsf{pk}(sk_{\mathcal{C}})}/_{y\_pk}\} \\
 & & \mid \{^{\mathsf{pair}(\mathsf{pk}(sk_M),k)}/_{e_1}\} \\
 & & \mid \{^{\mathsf{aenc}(\mathsf{pk}(sk_M),\mathsf{sign}(sk_{\mathcal{S}},k))}/_{x}\} \\
 & & \mid c(z).\text{if } \mathsf{fst}(\mathsf{sdec}(k,z)) = tag \text{ then} \\
 & & \overline{completedS}\langle \mathsf{pair}(k,\mathsf{eq}(\mathsf{pk}(sk_M),\mathsf{pk}(sk_{\mathcal{C}})))\rangle.Q \\
 & & \mid \{^{k}/_{e_2}\} \\
 & & \mid \{^{\mathsf{senc}(k,\mathsf{pair}(tag,s))}/_{z}\} \\
 & & \mid \{^{\mathsf{pair}(\mathsf{pk}(sk_{\mathcal{C}}),k)}/_{e_3}\}]
\end{array}
$$

Definition 7 (Validity of injective correspondence property) *Let E be an equational theory, and A_0 an extended process. We say that A_0* satisfies the injective correspondence property $\overline{f}\langle M\rangle \rightsquigarrow \mathsf{inj}\ \overline{g}\langle N\rangle$ *if for all execution paths*

$$A_0 \rightarrow^* \xrightarrow{\alpha_1} \rightarrow^* A_1 \rightarrow^* \xrightarrow{\alpha_2} \rightarrow^* \cdots \rightarrow^* \xrightarrow{\alpha_n} \rightarrow^* A_n,$$

there exists a partial injective function $h : \{1,\ldots,n\} \to \{1,\ldots,n\}$ such that for all $i \in \{1,\ldots,n\}$, substitution σ and variable e such that $\alpha_i = \nu\, e.\overline{f}\langle e\rangle$ and $e\varphi(A_i) =_E M\sigma$, then the following conditions are satisfied: (1) $h(i)$ is defined; (2) $\alpha_{h(i)} = \nu\, e'.\overline{g}\langle e'\rangle$ for some e' such that $e'\varphi(A_{h(i)}) =_E N\sigma$; and, (3) $h(i) < i$.

Returning to our Handshake protocol (Example 9), observe that authentication of $\mathcal{C}$ is injective; that is, if $\mathcal{S}$ reaches the end of the protocol with the belief that she has session key k with interlocutor $\mathcal{C}$, then she has indeed done so; and moreover, $\mathcal{C}$ has done so in this session (that is, there is an injective relationship).

Example 9 (Reasoning with correspondence, revisited) *Consider the annotated Handshake protocol and observe the injective correspondence property* $\overline{completedS}\langle \mathsf{pair}(y, \mathsf{true})\rangle \rightsquigarrow \mathsf{inj}\ \overline{startedC}\langle y\rangle$ *is valid. To show this conclusively, one has to consider all the possible execution paths, and show that for each of them, whenever there is a label* $\nu\, e.\overline{completedS}\langle e\rangle$ *associated with the substitution* $\{^{\mathsf{pair}(M,\mathsf{true})}/_{e}\}$*, then there is an earlier label* $\nu\, e'.\overline{startedC}\langle e'\rangle$ *and substitution* $\{^{M}/_{e'}\}$*; such that no two occurrences of the event* $completedS$ *are mapped to the same event* $startedC$*. ProVerif can be used to prove this result.*

Fixing the Handshake protocol. As mentioned in Section 1, the man-in-the-middle attack can be avoided by putting the participants' public keys along with k in the signature formed by $\mathcal{S}$. The resulting process and further discussion can be found in the extended version of this chapter [44].

4. Equivalence properties

The notion of indistinguishability is a powerful concept which allows us to reason about complex properties that cannot be expressed as secrecy or correspondence properties. Intuitively, two processes are said to be equivalent if an observer has no way to tell them apart. The processes may be handling different data, and internally performing quite different computations, but they look the same to an external observer. This notion allows us to define strong notions of secrecy and privacy.

A natural starting point for defining observational equivalence says that processes A and B are equivalent if they can output on the same channels, no matter what the context they are placed inside. Formally we write $A \Downarrow c$ when A can evolve to a process that can send a message on channel c, that is, when $A \rightarrow^* C[\overline{c}\langle M\rangle.P]$ for some term M, process P and evaluation context $C[\_]$ that does not bind c. This then allows us to characterise equivalence as follows: for all contexts $C[\_]$, we have $C[A] \Downarrow c$ if and only if $C[B] \Downarrow c$. One might think that this condition is too weak – it should say that $C[A]$ and $C[B]$ output the *same term* on the channel c – but that is not necessary, since if two processes output different terms then a context that inspects the output could be constructed to distinguish them.

4.1. Observational equivalence

In practice, the definition motivated above is hard to compute, and instead a stronger definition is often used. The stronger definition, called observational equivalence, has a recursive character. Roughly speaking, processes A and B are said to be observationally equivalent if they can output on the same channel for all contexts they are placed inside (as before); and also, for all contexts and every step made by A inside the context, there exists a step that B can make (inside the context), such that the resulting pair of processes are observationally equivalent (and vice-versa). We avoid the circularity of this informal characterisation, and define it rigorously as follows.

Definition 8 (Observational equivalence) *Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A\ \mathcal{R}\ B$ implies:*

1. *if $A \Downarrow c$, then $B \Downarrow c$;*
2. *if $A \to^* A'$, then $B \to^* B'$ and $A'\ \mathcal{R}\ B'$ for some B';*
3. *$C[A]\ \mathcal{R}\ C[B]$ for all closing evaluation contexts $C[\_]$.*

Note that we insist that $\mathcal{R}$ is symmetric, so we can state the Conditions 1 and 2 in a non-symmetric way (the symmetric part is guaranteed by the symmetry of $\mathcal{R}$). Note also the added condition that the two processes have the same domain; this captures the 'observable' difference between processes in which the value of a variable appearing in one frame, is not defined in the other.

A classical example illustrates the difference between observational equivalence and the notion of equivalence mentioned at the start of Section 4 (sometimes called *trace equivalence*). Intuitively, trace equivalence checks that the outputs on the traces allowed by $C[A]$ are the same as those of $C[B]$, but it does not enforce that *decisions* inside A and B occur at the same point. Let us use the notation $A_1 + A_2$ to mean the non-deterministic choice of A_1 or A_2. This is expressible in applied pi, as follows:

$$A_1 + A_2 \triangleq \nu\, a.(\overline{a}\langle \text{left}\rangle \mid \overline{a}\langle \text{right}\rangle \mid a(x).\text{if } x = \text{left then } A_1 \text{ else } A_2)$$

Here, left and right are names, and a scheduler chooses which message output $\overline{a}\langle\text{left}\rangle$ or $\overline{a}\langle\text{right}\rangle$ to perform, and hence whether A_1 or A_2 runs. Note that only one of A_1 and A_2 runs, in contrast with $A_1 \mid A_2$; moreover, only one of the outputs $\overline{a}\langle\text{left}\rangle$, $\overline{a}\langle\text{right}\rangle$ may occur, because there is only one input $a(x)$, and a is restricted. Now consider the processes $A \triangleq \overline{d}\langle b_1\rangle.\overline{d}\langle b_2\rangle + \overline{d}\langle b_1\rangle.\overline{d}\langle b_3\rangle$ and $B \triangleq \overline{d}\langle b_1\rangle.(\overline{d}\langle b_2\rangle + \overline{d}\langle b_3\rangle)$. One can see that they are trace equivalent, by experimenting with a variety of contexts. However, in A the decision between outputting b_2 or b_3 is taken earlier than it is in B. Observational equivalence captures this as an observable difference. Consider the context $C[\_] = \_ \mid d(y)$. Although $C[A]$ and $C[B]$ have the same output capabilities, $C[B]$ can evolve to a process (namely $\overline{d}\langle b_2\rangle + \overline{d}\langle b_3\rangle$) which is not equivalent to any process that $C[A]$ can evolve to (namely, the processes $\overline{d}\langle b_2\rangle$ or $\overline{d}\langle b_3\rangle$).

4.2. Labelled bisimilarity

The quantification over contexts makes the definition of observational equivalence hard to use in practice. Therefore, labelled bisimilarity is introduced, which is more suitable for both manual and automatic reasoning. Labelled bisimilarity relies on an equivalence relation between frames, called static equivalence, which we define first. Intuitively, two frames are statically equivalent if no 'test' $M = N$ can tell them apart, where M and N have variables that are substituted from the frame. Formally:

Definition 9 (Static equivalence) *Two closed frames φ and ψ are statically equivalent, denoted $\varphi \approx_s \psi$, if $\mathrm{dom}(\varphi) = \mathrm{dom}(\psi)$ and there exists a set of names $\tilde{n}$ and substitutions σ, τ such that $\varphi \equiv \nu\, \tilde{n}.\sigma$ and $\psi \equiv \nu\, \tilde{n}.\tau$ and for all terms M, N such that $\tilde{n} \cap (\mathrm{fn}(M) \cup \mathrm{fn}(N)) = \emptyset$, we have $M\sigma =_E N\sigma$ holds if and only if $M\tau =_E N\tau$ holds. We say two closed extended processes A, B are statically equivalent, and write $A \approx_s B$, when their frames are statically equivalent; that is, $\varphi(A) \approx_s \varphi(B)$.*

The relation is called *static* equivalence because it only examines the current state of the processes (as represented by their frames), and not the processes' dynamic behaviour (that is, the ways in which they may execute in the future). Thus, two processes are statically equivalent if they cannot be distinguished on the basis of their output so far. Static equivalence captures the static part of observational equivalence. More precisely, observational equivalence and static equivalence coincide on frames. Static equivalence can straightforwardly be shown to be closed under structural equivalence, internal reduction and application of closing evaluation contexts.

Example 10 (Static equivalence) *In the following examples we assume the signature* $\Sigma_S = \{\mathsf{hash}, \mathsf{fst}, \mathsf{snd}, \mathsf{pair}\}$ *and the smallest equational theory* E_S *satisfying the equations* $\mathsf{fst}(\mathsf{pair}(x, y)) = x$, $\mathsf{snd}(\mathsf{pair}(x, y)) = y$ *over all variables* x, y*, where as expected* hash*,* fst*,* snd *are unary functions and* pair *is a binary function.*

- $\nu\, m.\{^m/_x\} \approx_s \nu\, n.\{^n/_x\}$*; trivial, since they are structurally equivalent.*
- $\nu\, m.\{^m/_x\} \approx_s \nu\, n.\{^{\mathsf{hash}(n)}/_x\}$.
- $\{^m/_x\} \not\approx_s \{^{\mathsf{hash}(m)}/_x\}$*. The first one satisfies* $x = m$ *but the second one does not.*
- $\nu\, k.\{^k/_x\} \mid \nu\, s.\{^s/_y\} \approx_s \nu\, k.(\{^{\mathsf{hash}(\mathsf{pair}(a,k))}/_x\} \mid \{^{\mathsf{hash}(\mathsf{pair}(b,k))}/_y\}$.
- $\nu\, k.\{^k/_x\} \mid \nu\, s.\{^s/_y\} \not\approx_s \nu\, k.(\{^k/_x\} \mid \{^{\mathsf{hash}(k)}/_y\})$*, since the second one satisfies* $\mathsf{hash}(x) = y$ *and the first one does not.*
- $\nu\, s.\{^{\mathsf{pair}(s,s)}/_x\} \not\approx_s \nu\, s.\{^s/_x\}$*, since the first one satisfies* $\mathsf{pair}(\mathsf{fst}(x), \mathsf{snd}(x)) = x$ *but the second one does not.*

As mentioned, static equivalence captures the static part of observational equivalence. The following definition of labelled bisimilarity captures the dynamic part.

Definition 10 (Labelled bisimilarity) *Labelled bisimilarity* ($\approx_l$) *is the largest symmetric relation* $\mathcal{R}$ *on closed extended processes such that* $A\ \mathcal{R}\ B$ *implies:*

1. $A \approx_s B$*;*
2. *if* $A \to A'$*, then* $B \to^* B'$ *and* $A'\ \mathcal{R}\ B'$ *for some* B'*;*
3. *if* $A \xrightarrow{\alpha} A'$ *and* $\mathrm{fv}(\alpha) \subseteq \mathrm{dom}(A)$ *and* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset$*; then* $B \to^* \xrightarrow{\alpha} \to^* B'$ *and* $A'\ \mathcal{R}\ B'$ *for some* B'*.*

Clauses 2 and 3 of this definition correspond to classical notions of bisimilarity [42]. Notice the use of the "largest relation" construction, to allow us to insist that the processes A' and B' are again within the relation. Clause 1 asserts static equivalence at each step of the bisimulation.

Let us now consider the side condition of Clause 3. Recall that there are three possibilities for α, namely $c(M)$, $\overline{c}\langle u\rangle$, and $\nu u.\overline{c}\langle u\rangle$, where u is either a channel name or a variable of base type. In Clause 3, α can have free variables (in the cases $c(M)$ and $\overline{c}\langle u\rangle$), but any such variables must be defined by the frame of A (and since $A \approx_s B$, also by the frame of B). Hence the condition $\mathrm{fv}(\alpha) \subseteq \mathrm{dom}(A)$. The label α can also have bound names (in the case $\nu d.\overline{c}\langle d\rangle$, where d is a channel name), in which case the transition $A \xrightarrow{\alpha} A'$ has the effect of removing the restriction on a bound channel name. That channel must not occur free in B, to avoid confusing the restricted name with the global name; hence the condition $\mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset$. To see this in more detail, consider A, B such that $A \approx_l B$, and suppose $a \in \mathrm{fn}(B) \backslash \mathrm{fn}(A)$. (For example, A is 0 and B is if $a =$

b then $\overline{c}\langle a\rangle$.) Intuitively, we would like $A \mid (\nu a.\overline{c}\langle a\rangle) \approx_l B \mid (\nu a.\overline{c}\langle a\rangle)$. But to achieve that, we need the side condition, for otherwise the transition $A \mid (\nu a.\overline{c}\langle a\rangle) \xrightarrow{\nu a.\overline{c}\langle a\rangle} A$ would have to be matched on the right hand side by $B \mid (\nu a.\overline{c}\langle a\rangle) \xrightarrow{\nu a.\overline{c}\langle a\rangle} B$, which is false.

Working with labelled bisimilarity. To show labelled bisimilarity of A and B, it is necessary to find a symmetric relation $\mathcal{R}$ that satisfies the conditions contained in Definition 10, such that $A \mathcal{R} B$. Note that the $\mathcal{R}$ we find is not required to be the largest one. That is because the set of relations that satisfies the conditions contained in Definition 10 is closed under union. Therefore, the largest one is the union of them all. *Any* relation satisfying the conditions of Definition 10 can serve as a witness to show that two processes are labelled bisimilar. Note also that, although labelled bisimilarity is easily seen to be an equivalence relation, the relation $\mathcal{R}$ is merely required to be symmetric.

Example 11 (Labelled bisimilarity) *We prove that for all closed processes P_1, P_2, C and names $\tilde{n}$ such that $c \in \tilde{n}$, we have $\nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C \approx_l \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C$. Intuitively, the equivalence holds because the only choice that the left side has, which the right side does not, is an internal reduction (private channel communication). Let $A \mathcal{R} B$ hold if there exists a closed extended process C' such that one of the following holds:*

- *$A \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C'$ and $B \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$; or*
- *$B \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C'$ and $A \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$; or*
- *$A \equiv B$.*

One may show that $\mathcal{R}$ satisfies the conditions contained in Definition 10. First, $\mathcal{R}$ is easily seen to be symmetric. For the next part, suppose $A \mathcal{R} B$.

1. *We show $A \approx_s B$. There are three cases.*
   (a) *$A \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C'$ and $B \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$. Since $\nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \approx_s \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\})$, we have $A \approx_s B$.*
   (b) *$B \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C'$ and $A \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$. In this case, a similar argument applies.*
   (c) *$A \equiv B$ and hence we trivially have $A \approx_s B$.*
2. *Now suppose $A \to A'$. There are again three cases to consider.*
   (a) *$A \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C'$ and $B \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$. If $A' \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$, then let $B' \triangleq B$. Otherwise, $A' \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C''$ for some C'', in which case let $B' \triangleq \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C''$.*
   (b) *$A \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$ and $B \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C'$. Since $B \to A$, it follows that $B \to^* A'$, so let $B' \triangleq A'$.*
   (c) *$A \equiv B$ and hence the result follows trivially by $B' \triangleq A'$.*

   In all cases we have $A' \mathcal{R} B'$, as required.
3. *Now suppose $A \xrightarrow{\alpha} A'$. Again we have three cases:*
   (a) *$A \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C'$ and $B \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$. Since only the C' part is able to make a labelled transition, $A' \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C''$ for some C'', in which case let $B' \triangleq \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C''$, and we have $B \xrightarrow{\alpha} B'$.*

(b) $A \equiv \nu\tilde{n}.(P_1 \mid P_2\{^a/_x\}) \mid C'$ *and* $B \equiv \nu\tilde{n}.(\overline{c}\langle a\rangle.P_1 \mid c(x).P_2) \mid C'$. *Since* $B \to A$, *it follows that* $B \to^* \xrightarrow{\alpha} A'$, *so let* $B' \triangleq A'$.
(c) $A \equiv B$ *and hence the result is trivial by* $B' \triangleq A'$.

In all cases we have $A' \mathcal{R} B'$, *as required.*

Abadi & Fournet [8] state the following useful results, although as far as we are aware fully detailed proofs have not yet been published.

Lemma 1 *Given closed extended processes* A, B *and a closing evaluation context* $C[\_]$, *we have* $A \approx_l B$ *implies* $C[A] \approx_l C[B]$.

Theorem 1 *Observational equivalence and labelled bisimilarity coincide:* $\approx = \approx_l$.

The condition stating that active substitutions are of base type (Section 2.1) is crucial to Theorem 1. Without this constraint, one would have the following counterexample: $\nu\, a.(\{^a/_x\} \mid x(y).\overline{c}\langle n\rangle) \not\approx \nu\, a.\{^a/_x\}$ (this can be seen using the closing evaluation context $C[\_] = \overline{x}\langle b\rangle \mid \_$); whereas $\nu\, a.(\{^a/_x\} \mid x(y).\overline{c}\langle n\rangle) \approx_l \nu\, a.\{^a/_x\}$.

4.3. Strong secrecy

Our earlier definition of secrecy required that the secret was not obtained by the adversary. We will now use notions of equivalence to demonstrate a stronger notion of secrecy which states an adversary is unable to distinguish when the secret changes [1,30]. Intuitively, this means the value of the secret should not effect the observable behaviour of the protocol. Strong secrecy is useful, for example, to capture the adversary's inability to learn any partial information about the secret.

Strong secrecy is also useful to formalise 'dictionary attacks', also known as 'guessing attacks', in which the attacker tries all the possible values of the secret until he finds the right one. If a secret is chosen from a relatively small set of possible values, then such an attack might be possible; we call such a secret "weak". Passwords are often examples of weak secrets. For example, a server which requires the user to send her password s, deterministically encrypted with server's public key pk, that is, to send $\{\!|s|\!\}^{\mathsf{a}}_{pk}$, is vulnerable to guessing attacks on s. That is because an attacker in possession of $\{\!|s|\!\}^{\mathsf{a}}_{pk}$ can make a guess s' of the password s, and check the validity of his guess by constructing $\{\!|s'|\!\}^{\mathsf{a}}_{pk}$ and comparing it with $\{\!|s|\!\}^{\mathsf{a}}_{pk}$. Suppose instead the server requires the user to include a random nonce in the encryption, that is, to send $\{\!|\langle s, r\rangle|\!\}^{\mathsf{a}}_{pk}$, where r is a nonce chosen from a large set. In that case the system is not vulnerable to guessing attacks.

This may be formalised by the following definitions [14,32].

Definition 11 *Let* $\varphi \equiv \nu n.\varphi'$ *be a frame. We say that* φ *is* resistant to guessing attacks *against* n *if, and only if,* $\nu n.(\varphi' \mid \{^n/_x\}) \approx_s \nu n'.\nu n.(\varphi' \mid \{^{n'}/_x\})$ *where* n' *is a fresh name and* x *is a variable such that* $x \notin \mathrm{dom}(\varphi)$.

The definition says that the environment cannot distinguish between a situation in which he possesses the correct value of the secret, from another one in which he has some random value instead. This definition can be extended to the more general case.

Definition 12 (Guessing attacks) *Let A be a process and $n \in \mathrm{bn}(A)$. We say that A is* resistant to guessing attacks *against n if, for every process B such that $A(\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^* B$, then we have that $\varphi(B)$ is resistant to guessing attacks against n.*

Example 12 (TPM authentication [26]) *We consider a protocol where a client $\mathcal{A}$ shares a secret s with a server $\mathcal{B}$ (the protocol is based on the one used in the Trusted Platform Module [48]). In this example, the terms M and N are commonly used for tuples, so it is convenient to introduce a little syntactic sugar to express them. We write $(\!|M_1, \ldots, M_n|\!)$ to mean $\mathsf{pair}(M_1, \mathsf{pair}(M_2, \mathsf{pair}(\ldots, \mathsf{pair}(M_n, *)\ldots)))$, where $*$ is any constant. We also write $\mathsf{1st}(M)$, $\mathsf{2nd}(M)$, and $\mathsf{3rd}(M)$ to mean $\mathrm{fst}(M)$, $\mathrm{fst}(\mathrm{snd}(M))$, and $\mathrm{fst}(\mathrm{snd}(\mathrm{snd}(M)))$ respectively. Note that $(\!|M_1, \ldots, M_k|\!)$, $\mathsf{1st}(N)$, etc., are not new functions, but merely syntactic sugar for combinations of functions that we already have. The protocol proceeds as follows. The client sends commands to the server, and authenticates them by supplying a MAC of the command values, keyed on s. An additional nonce n is supplied and used in the MAC, to ensure it has high entropy. More precisely, to run the command 'comm', $\mathcal{A}$ sends the message $(\!|\mathit{comm}, n, \mathsf{mac}(s, (\!|\mathit{comm}, n|\!))|\!)$. $\mathcal{B}$ checks the MAC by reconstructing it, and if it is correct, sends the response 'resp'. This protocol may be modelled by the following process P, where we assume the signature $\Sigma_T = \Sigma_S \cup \{\mathsf{mac}\}$ and equational theory E_S (recall that Σ_S and E_S were defined in Example 10, and note that binary function* mac *has no equations).*

$$P \triangleq \nu\, s.(!P_A \mid !P_B)$$

$$P_A \triangleq \nu\, n.\overline{c}\langle(\!|\mathit{comm},\ n,\ \mathsf{mac}(s, (\!|\mathit{comm}, n|\!))|\!)\rangle$$

$$P_B \triangleq c(x).\text{if } \mathsf{3rd}(x) = \mathsf{mac}(s, (\!|\mathsf{1st}(x), \mathsf{2nd}(x)|\!)) \text{ then } \overline{c}\langle \mathit{resp}\rangle$$

P is vulnerable to guessing attacks on s. To see this, we consider the transition

$$P \xrightarrow{\nu x.\overline{c}\langle x\rangle} \nu s.(\nu n.\{(\!|\mathrm{comm},\ n,\ \mathsf{mac}(s, (\!|\mathrm{comm}, n|\!))|\!)/_x\} \mid !P_A \mid !P_B)$$

The frame of this latter process is vulnerable to guessing attacks on s, since we have

$$\begin{aligned}&\nu s.\nu n.(\{(\!|\mathrm{comm},\ n,\ \mathsf{mac}(s, (\!|\mathrm{comm}, n|\!))|\!)/_x\} \mid \{s/_z\})\\ &\quad \not\approx_s \nu s'.\nu s.\nu n.(\{(\!|\mathrm{comm},\ n,\ \mathsf{mac}(s, (\!|\mathrm{comm}, n|\!))|\!)/_x\} \mid \{s'/_z\})\end{aligned}$$

as witnessed by the test

$$\mathsf{3rd}(x) = \mathsf{mac}(z, (\!|\mathsf{1st}(x), \mathsf{2nd}(x)|\!)).$$

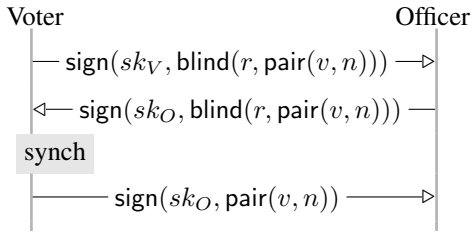
A comparison of two views of secrecy. The reachability-based secrecy introduced in Section 3.1 requires that the adversary should never learn secret values; the equivalence-based (strong) secrecy introduced in this section states that two executions of a protocol are indistinguishable regardless of the chosen secret values. The formulation of strong secrecy offers a higher degree of security, but reachability-based secrecy has been more widely studied resulting in more automated support. For further discussion on the relationship between these two styles of secrecy the reader is referred to [30,1], see also Blanchet for a more generic formalisation of strong secrecy [20].

4.4. Vote privacy for electronic voting

Electronic voting systems are being introduced, or trialled, in several countries to provide more efficient voting procedures with an increased level of security. The applied pi calculus can be used to formalise the properties which an electronic voting scheme should satisfy [39,34,12]. For example, the property of vote privacy (or ballot secrecy) means that no-one (not even the election officers) can link a particular vote with the voter.

Consider the following protocol, which is a simplification of the one by Fujioka, Okamoto, and Ohta [37]. We assume the signature $\Sigma_F = \{\mathsf{true}, \mathsf{fst}, \mathsf{snd}, \mathsf{pk}, \mathsf{getmsg}, \mathsf{pair}, \mathsf{sign}, \mathsf{checksign}, \mathsf{blind}, \mathsf{unblind}\}$, the arities of which are defined in the usual way; that is, true is a constant; $\mathsf{fst}, \mathsf{snd}, \mathsf{pk}, \mathsf{getmsg}$ are unary functions; and $\mathsf{pair}, \mathsf{sign}$, $\mathsf{checksign}, \mathsf{blind}, \mathsf{unblind}$ are binary functions. The protocol [37] relies on *blind signatures*; with this cryptographic primitive, an election officer can sign a text without having seen it. The voter first blinds the text and the officer signs it. The voter can then unblind the message to recover the officer's signature on the text. We do not need to consider how this cryptography actually works; we can encode the effect using the equation $\mathsf{unblind}(x, \mathsf{sign}(y, \mathsf{blind}(x, z))) = \mathsf{sign}(y, z)$ which says that if one blinds a text z using a blinding factor x, and then signs it with the private key y, and then unblinds it (again using x), then the result is the text z signed by y in the usual way. Blind signatures are useful in electronic voting because they allow the officer to sign a vote (thus establishing its eligibility for counting) without knowing what the vote is. The equations associated with the remaining functions in Σ_F are defined in the usual way (see Section 2.1.1).

The protocol proceeds as follows. The voter wishing to vote for candidate v creates a nonce n, and blinds the tuple $\langle v, n\rangle$ using a random blinding factor r. She then signs this value and sends it to the election officer. The officer checks the voter's eligibility and also checks she has not already voted. If these checks succeed, then the officer signs the blinded vote-nonce pair and sends it back to the voter. The voter can now unblind the signature, recovering the officer's signature on the vote-nonce pair. Once all voters have obtained the officer's signature in this way, they can anonymously submit the signed vote-nonce pair for counting. The protocol thus proceeds as follows:



The purpose of the value n is to allow the officer to count the signed vote only once. The synchronisation point, denoted synch, ensures that every voter receives the officer's signature before any of them submit the unblinded value for counting. It is necessary to ensure vote privacy; without it, traffic analysis might link the voter's signature in the first message with the anonymous submission in the third one. This protocol is inadequate for a real election, but it does satisfy the property of vote privacy.

We formalise this illustrative voting protocol in the applied pi calculus as follows. We assume a public channel c for communication between the voter and the officer, and a channel $vote$ for the officer to declare the list of counted votes.

$$
\begin{aligned}
P \;\triangleq\; & \nu\, sk_1 \;\ldots\; \nu\, sk_n.\; \nu\, sk_O. \\
& \text{let } pk_1 = \mathsf{pk}(sk_1) \text{ in } \ldots \text{ let } pk_n = \mathsf{pk}(sk_n) \text{ in } \overline{c}\langle pk_1\rangle.\ldots.\overline{c}\langle pk_n\rangle. \\
& \text{let } pk_O = \mathsf{pk}(sk_O) \text{ in } \overline{c}\langle pk_O\rangle. \\
& (P_V\{^{sk_1}/_{sk_V}, {}^{v_1}/_{v}\} \mid \cdots \mid P_V\{^{sk_n}/_{sk_V}, {}^{v_n}/_{v}\} \mid\; !P_O \mid S)
\end{aligned}
$$

$$
\begin{aligned}
P_V \;\triangleq\; & \nu n.\nu r.\text{let } bvn = \mathsf{blind}(r, \mathsf{pair}(v, n)) \text{ in} \\
& \overline{c}\langle \mathsf{pair}(\mathsf{pk}(sk_V), \mathsf{sign}(sk_V, bvn))\rangle. \\
& c(x).\text{if } \mathsf{checksign}(pk_O, x) = \mathsf{true} \text{ then} \\
& \text{if } \mathsf{getmsg}(x) = bvn \text{ then} \\
& \textit{synch.} \\
& \overline{c}\langle \mathsf{unblind}(r, x)\rangle
\end{aligned}
$$

$$
\begin{aligned}
P_O \;\triangleq\; & c(y).\text{if } \mathsf{checksign}(\mathsf{fst}(y), \mathsf{snd}(y)) = \mathsf{true} \text{ then} \\
& \text{if } \mathsf{Eligible}(\mathsf{fst}(y)) = \mathsf{true} \text{ then} \\
& \overline{c}\langle \mathsf{sign}(sk_O, \mathsf{getmsg}(\mathsf{snd}(y)))\rangle. \\
& c(w). \\
& \text{if } \mathsf{checksign}(sk_O, w) = \mathsf{true} \text{ then} \\
& \text{if } \mathsf{NotSeen}(w) = \mathsf{true} \text{ then} \\
& \overline{vote}\langle \mathsf{fst}(\mathsf{getmsg}(w))\rangle
\end{aligned}
$$

In the above process, we assume functions $\mathsf{Eligible}(x)$ and $\mathsf{NotSeen}(x)$ are available to the officer. $\mathsf{Eligible}(x)$ checks whether the voter with public key x is eligible and hasn't already voted; $\mathsf{NotSeen}(x)$ stores x and returns true if it has not seen x before; otherwise it returns false. The purpose of $\mathsf{NotSeen}(x)$ is to ensure that the voter can't use the signed token twice. We also assume a subprocess *synch* which a voter uses to synchronise with other voters. This can be modelled by defining a process S to coordinate the synchronisation. In the case of n voters, the coordination process receives precisely n tokens, and then sends them back. To synchronise with the other voters, a voter sends a token to this synchronisation coordinator and awaits its response. Thus,

$$
\begin{aligned}
\text{synch} &\triangleq \overline{syn}\langle *\rangle.syn'(o) \\
S &\triangleq syn(x_1) \ldots syn(x_n).\overline{syn'}\langle *\rangle \ldots \overline{syn'}\langle *\rangle
\end{aligned}
$$

where syn, syn' are private channels shared between the voters and S, $*$ is any name and o is a variable.

Now we define the property of vote privacy. Since it should hold even if the officer is corrupt, we can suppose that the officer is part of the attacker, and we need not consider its formalisation as P_O above. This means the secrecy of the officer's key is not required for the vote privacy property, so we can treat it as a free name. Moreover, the voters' secret keys are also not required to be kept secret in order to ensure the vote privacy property. They too can be treated as free names. (Of course, these keys *are* required to be secret for other properties.)

Consider two voters $\mathcal{A}$, $\mathcal{B}$ and two candidates v_a, v_b. Based upon [34], we formalise vote privacy for two voters with the assertion that the attacker (including the election officers) cannot distinguish between a situation in which $\mathcal{A}$ votes v_a and $\mathcal{B}$ votes v_b, from another one in which $\mathcal{A}$ votes v_b and $\mathcal{B}$ votes v_a. We will write this as the equivalence:

$$\nu\, syn.\nu\, syn'.(P_V\{^{sk_A}/_{sk_V}, ^{v_a}/_v\} \mid P_V\{^{sk_B}/_{sk_V}, ^{v_b}/_v\} \mid S)$$
$$\approx_l \nu\, syn.\nu\, syn'.(P_V\{^{sk_A}/_{sk_V}, ^{v_b}/_v\} \mid P_V\{^{sk_B}/_{sk_V}, ^{v_a}/_v\} \mid S)$$

To prove this equivalence, let us call the left hand side LHS, and the right hand side RHS. We need to show a relation $\mathcal{R}$ satisfying the requirements of Definition 10 such that LHS $\mathcal{R}$ RHS. As we have seen, the idea of $\mathcal{R}$ is to relate successors of LHS and RHS that are expected to correspond to each other in the bisimulation. As LHS evolves, the corresponding evolution of RHS is the one that mimics $\mathcal{A}$ moves in LHS with $\mathcal{A}$ moves in RHS, and $\mathcal{B}$ moves in LHS with $\mathcal{B}$ moves in RHS, up to the synchronisation. After the synchronisation, $\mathcal{A}$ moves in LHS are mimicked by $\mathcal{B}$ moves in RHS, and $\mathcal{B}$ moves in LHS are mimicked by $\mathcal{A}$ moves in RHS. The reason for the 'swap' in mimicking is to ensure that the static equivalences will hold; before the synchronisation, the output data produced by $\mathcal{A}$ on both the LHS and RHS are indistinguishable (and similarly for $\mathcal{B}$). Observe that the output data reveals the keys $\mathsf{pk}(sk_A)$, $\mathsf{pk}(sk_B)$ (matched by $\mathsf{pk}(sk_A)$, $\mathsf{pk}(sk_B)$) and the signed blind votes $\mathsf{sign}(sk_A, \mathsf{blind}(r, \mathsf{pair}(v_a, n)))$, $\mathsf{sign}(sk_B, \mathsf{blind}(r', \mathsf{pair}(v_b, n')))$ (matched by $\mathsf{sign}(sk_A, \mathsf{blind}(r, \mathsf{pair}(v_b, n)))$, $\mathsf{sign}(sk_B, \mathsf{blind}(r', \mathsf{pair}(v_a, n'))))$ where names n, n', r, r' are under restriction in both LHS and RHS. Indistinguishability between the LHS and RHS with respect to the signed blind votes is due to the properties of blinding. After the synchronisation $\mathcal{A}$ will reveal v_a on the LHS and v_b on the RHS (similarly $\mathcal{B}$ will reveal v_b on the LHS and v_a on the RHS). Hence after synchronisation the actions mimicked are swapped. A formal proof of this result is included in the extended version of this chapter [44].

5. Outlook

This chapter studies the applied pi calculus and demonstrates its applicability for analysing secrecy, correspondence and observational equivalence properties in the context of security protocols. In this section we briefly refer to some on-going research about the calculus. Our discussion is neither exhaustive nor complete.

The ability to reason with security properties is largely dependent on being able to reason with the equational theory. Abadi & Cortier [6] have studied decidability of secrecy and static equivalence for a large class of equational theories in the presence of a passive adversary. Subsequent work by Baudet, Cortier & Delaune [15] and Ciobâcă, Delaune & Kremer [28] introduces automated tools for analysing static equivalence with respect to convergent equational theories.

For an active adversary, reachability properties, in particular secrecy, have been studied in [18,2] with updated and extended versions [3] and in Chapter *"Using Horn clauses for analyzing protocols"* of this book. Strong secrecy has been considered in [20]. Correspondence properties have been presented in the context of authenticity [19], and subsequently extended upon and revised in [21]. Proofs of observational equivalence have been studied with respect to processes that differ only by the choice of the terms that they contain, using the notion of uniformity [22,23]. But this notion of uniformity is often too strict; for example, the voting process discussed in this chapter does not satisfy it. A practical approach to overcoming the problem is introduced [35], [45, Chapter 5] and subsequently used [12,31]. As an alternative to uniformity as a proof technique, a symbolic version of the applied pi calculus is introduced in [33]; by treating inputs symboli-

cally, it avoids potentially infinite branching of processes due to inputs from the environment. Cortier & Delaune [29] have shown that observational equivalence coincides with trace equivalence for determinate processes; and based on the symbolic semantics, this yields a decision procedure for observational equivalence of finite determinate processes without replication.

Acknowledgements

We are very grateful to Myrto Arapinis and the editors of the Handbook for careful reading of a draft of this chapter. This work has been partly supported by the EPSRC projects: *UbiVal: Fundamental Approaches to Validation of Ubiquitous Computing Applications and Infrastructures* (EP/D076625/2); and *Verifying Interoperability Requirements in Pervasive Systems* (EP/F033540/1). This work was partly done while Ben Smyth was at CNRS, Département d'Informatique, École Normale Supérieure, Paris, France with support from the *Direction Générale pour l'Armement* (DGA).

References

[1] Martín Abadi. Security Protocols and their Properties. In Friedrich L. Bauer and Ralf Steinbrüggen, editors, *Foundations of Secure Computation*, NATO Science Series, pages 39–60. IOS Press, 2000.

[2] Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *POPL'02: Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 33–44. ACM Press, 2002.

[3] Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, 2005.

[4] Martín Abadi and Bruno Blanchet. Computer-Assisted Verification of a Protocol for Certified Email. *Science of Computer Programming*, 58(1–2):3–27, 2005. Special issue SAS'03.

[5] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just Fast Keying in the Pi Calculus. *ACM Transactions on Information and System Security*, 10(3):1–59, 2007.

[6] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. In *ICALP'04: Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, volume 3142 of *LNCS*, pages 46–58. Springer, 2004.

[7] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1–2):2–32, 2006.

[8] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL'01: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115. ACM Press, 2001.

[9] Martín Abadi and Cédric Fournet. Private email communication, 24th May 2006.

[10] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *CCS'97: 4th ACM conference on Computer and Communications Cecurity*, pages 36–47. ACM Press, 1997.

[11] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[12] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-Calculus. In *CSF'08: 21st IEEE Computer Security Foundations Symposium*, pages 195–209. IEEE Computer Society, 2008.

[13] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-Knowledge in the Applied Pi-calculus and Automated Verification of the Direct Anonymous Attestation Protocol. In *S&P'08: 29th IEEE Symposium on Security and Privacy*, pages 202–215. IEEE Computer Society, 2008.

[14] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *CCS'05: Proceedings of the 12th Conference on Computer and Communications Security*, pages 16–25. ACM Press, 2005.

[15] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. In *RTA'09: Proceedings of the 20th International Conference on Rewriting Techniques and Applications*, volume 5595 of *LNCS*, pages 148–163. Springer, 2009.

[16] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile Processes, Nominal Data, and Logic. In *LICS '09: 24th Annual IEEE Symposium on Logic In Computer Science*, pages 39–48. IEEE Computer Society, 2009.

[17] Karthikeyan Bhargavan, Cedric Fournet, Andrew D. Gordon, and Stephen Tse. Verified Interoperable Implementations of Security Protocols. In *CSFW'06: IEEE Computer Security Foundations Workshop*, pages 139–152. IEEE Computer Society, 2006.

[18] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW'01: Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 82–96. IEEE Computer Society, 2001.

[19] Bruno Blanchet. From Secrecy to Authenticity in Security Protocols. In *SAS'02: Proceedings of the 9th International Static Analysis Symposium*, volume 2477 of *LNCS*, pages 342–359. Springer, 2002.

[20] Bruno Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *S&P'04: 25th IEEE Symposium on Security and Privacy*, pages 86–100. IEEE Computer Society, 2004.

[21] Bruno Blanchet. Automatic Verification of Correspondences for Security Protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[22] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *LICS'05: Proceedings of the 20th IEEE Symposium on Logic in Computer Science*, pages 331–340. IEEE Computer Society, 2005.

[23] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[24] Bruno Blanchet and Avik Chaudhuri. Automated Formal Analysis of a Protocol for Secure File Sharing on Untrusted Storage. In *S&P'08: 29th IEEE Symposium on Security and Privacy*, pages 417–431. IEEE Computer Society, 2008.

[25] Bruno Blanchet and Ben Smyth. ProVerif: Automatic Cryptographic Protocol Verifier User Manual & Tutorial. Available from `http://www.proverif.ens.fr/`, 2010.

[26] Liqun Chen and Mark D. Ryan. Offline dictionary attack on TCG TPM weak authorisation data, and solution. In David Gawrock, Helmut Reimer, Ahmad-Reza Sadeghi, and Claire Vishik, editors, *Future of Trust in Computing*. Vieweg & Teubner, 2008.

[27] Liqun Chen and Mark D. Ryan. Attack, solution and verification for shared authorisation data in TCG TPM. In *FAST'09: 6th Formal Aspects in Security and Trust*, LNCS. Springer, 2009.

[28] Ştefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. In *CADE'09: Proceedings of the 22nd International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 355–370. Springer, 2009.

[29] Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *CSF'09: 22nd IEEE Computer Security Foundations Symposium*, pages 266–276. IEEE Computer Society, 2009.

[30] Véronique Cortier, Michaël Rusinowitch, and Eugen Zălinescu. Relating two standard notions of secrecy. *Logical Methods in Computer Science*, 3(3), 2007.

[31] Morten Dahl, Stéphanie Delaune, and Graham Steel. Formal Analysis of Privacy for Vehicular Mix-Zones. In *ESORICS'10: 15th European Symposium on Research in Computer Security*, volume 6345 of *LNCS*, pages 55–70. Springer, 2010.

[32] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Composition of password-based protocols. In *CSF'08: Proceedings of the 21st IEEE Computer Security Foundations Symposium*, pages 239–251, Pittsburgh, PA, USA, 2008. IEEE Computer Society.

[33] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security*, 2009.

[34] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying Privacy-type Properties of Electronic Voting Protocols. *Journal of Computer Security*, 2009.

[35] Stéphanie Delaune, Mark D. Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi-calculus. In *IFIPTM'08: Proceedings of the 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security*, volume 263 of *IFIP Conference Proceedings*, pages 263–278. Springer, 2008.

[36] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory*, 29:198–208, 1983.

[37] Atsushi Fujioka, Tatsuaki Okamoto, and Kazui Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT'92: Advances in Cryptology*, volume 718 of *LNCS*, pages 244–251. Springer, 1992.

[38] Steve Kremer and Mark D. Ryan. Analysis of an Electronic Voting Protocol in the Applied Pi-Calculus. In *ESOP'05: Proceedings of the 14th European Symposium On Programming*, volume 3444 of *LNCS*, pages 186–200. Springer, 2005.

[39] Steve Kremer, Mark D. Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *ESORICS'10: 15th European Symposium on Research in Computer Security*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010. An extended version of this paper appears in [45, Chapter 3].

[40] Ralf Küsters and Tomasz Truderung. An Epistemic Approach to Coercion-Resistance for Electronic Voting Protocols. In *S&P'09: 30th IEEE Symposium on Security and Privacy*, pages 251–266. IEEE Computer Society, 2009.

[41] Gavin Lowe. A hierarchy of authentication specifications. In *CSFW'97: 10th IEEE workshop on Computer Security Foundations*, pages 31–43. IEEE Computer Society, 1997.

[42] Robin Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[43] Aybek Mukhamedov, Andrew D. Gordon, and Mark D. Ryan. Towards a Verified Reference Implementation of the Trusted Platform Module. In *17th International Workshop on Security Protocols*, LNCS. Springer, 2009.

[44] Mark D. Ryan and Ben Smyth. Applied pi calculus. Available from `http://www.bensmyth.com/publications/10applied-pi/`, 2010.

[45] Ben Smyth. *Formal verification of cryptographic protocols with automated reasoning*. PhD thesis, School of Computer Science, University of Birmingham, 2011.

[46] Ben Smyth, Mark D. Ryan, and Liqun Chen. 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*, volume 4572 of *LNCS*, pages 218–231, 2007. An extended version of this paper appears in [45, Chapter 4].

[47] D. Syme, A. Granicz, and A. Cisternino. *Expert F#*. Apress, 2007.

[48] Trusted Computing Group. TPM Specification version 1.2. Parts 1–3., 2007. Available from `http://www.trustedcomputinggroup.org/specs/TPM/`.