Secrecy-preserving Refinement

Jan Jürjens*

University of Oxford, GB

This is the web-version 9/3/01 of a paper at Formal Methods Europe 2001, LNCS, Springer Verlag

Abstract. A useful paradigm of system development is that of *stepwise refinement*. In contrast to other system properties, many security properties proposed in the literature are not preserved under refinement (*refinement paradox*).
We present work towards a framework for stepwise development of secure systems by showing a notion of secrecy (that follows a standard approach) to be preserved by standard refinement operators in the specification framework Focus (extended with cryptographic primitives). We also give a rely/guarantee version of the secrecy property and show preservation by refinement. We use the secrecy property to uncover a previously unpublished flaw in a proposed variant of TLS, propose a correction and prove it secure. We give an abstract specification of a secure channel satisfying secrecy and refine it to a more concrete specification that by the preservation result thus also satisfies secrecy.

1 Introduction

A useful paradigm of system development is that of *stepwise refinement*: One starts with an abstract specification and refines it in several steps to a concrete specification which is implemented. Advantage of this approach is that mistakes may be detected rather early in the development cycle, which leads to considerable savings (late correction of requirements errors costs up to 200 times as much as early correction [Boe81]).

Clearly, the concrete specification must have all relevant properties of the initial specification. This is indeed the case for system properties that can be expressed as properties on traces (taking refinement to be reverse inclusion on trace sets). A classical example is the Alpern-Schneider framework of safety and liveness properties.

However, many security properties proposed in the literature are properties on trace sets rather than traces and give rise to the *refinement paradox* which means that these properties are not preserved under refinement (for noninterference this is pointed out in [McL94, McL96]; the same observation applies to equivalence-based notions of secrecy explained e. g. in [Aba00]).

* Internet: `http://www.jurjens.de/jan` – e-mail: `jan@comlab.ox.ac.uk` – This work was supported by the Studienstiftung des deutschen Volkes and Lucent Technologies.

For such properties, developing secure systems in a stepwise manner requires to redo security proofs at each refinement step. More worryingly, since an implementation is necessarily a refinement of its specification, an implementation of a secure specification may not be secure. Thus the results of verifying such properties on the level of specifications needs to be applied with care, as pointed out in [RS98].

In this work, we seek to address this problem. In the specification framework Focus [Bro99, BS00] (extended by cryptographic operations including symmetric and asymmetric encryption and signing) we consider a secrecy property following the approach of [DY83] and show that it is preserved by the various refinements of the framework. We also give a rely/guarantee version of the secrecy property and show preservation by refinement. We demonstrate adequacy of the proposed secrecy notion by using it to uncover a previously unpublished flaw in a variant of the handshake protocol of TLS[1] proposed in [APS99], to propose a correction and to prove it secure. As an example for the stepwise development of a secure system we then give an abstract specification of a secure channel and refine it to a more concrete specification. The abstract specification satisfies secrecy, and by our preservation result the concrete one does as well.

In the next subsection we put our work into context and refer to related work on the subject of this paper. In Section 2, we introduce the specification framework Focus with the cryptographic extension. In Section 3 we give the secrecy properties considered in this work. In Section 4 we define the notions of refinement provided in Focus and show that they preserve the secrecy properties. In Section 5 we specify the variant of the TLS handshake protocol in our language, demonstrate the flaw, give a corrected version and prove it secure. In Section 6 we develop a specification of a secure channel in a stepwise manner. After that, we conclude.

Some of the proofs have to be omitted for lack of space; they are to be found in the long version of this paper to be published.

1.1 Security and Refinement

In the specification of systems one may employ nondeterminism in different ways, including the following:

under-specification: to simplify design or verification of systems. Certain details may be left open in the early phases of system development

[1] TLS is the successor of the Internet security protocol SSL.

or when verifying system properties, to simplify matters or because they may not be known (for example the particular scheduler used to resolve concurrency).

unpredictability: to provide security. For example, keys or nonces are chosen in a way that should make them unguessable.

While the first kind of nondeterminism is merely a tool during development or verification, the second is a vital part of the functionality of a system. When one identifies the two kinds of nondeterminism one faces the refinement paradox mentioned above.

We separate the two kinds of nondeterminism in the following way: The nondeterminism of functional importance for the system is *only* modelled by specific primitives (such as key generation), possibly making use of syntax. Thus the security of a system does *not* rely on nondeterministic choice in the formal model. Providing unpredictability through underspecification may be compared to providing *security by obscurity.* It has been argued that this is inferior to open design [SS75].

It is quite common in the formal modelling of security to provide special primitives for operations such as key generation, encryption etc.. However, security properties for nondeterministic specifications often also use the nondeterministic choice operators to provide unpredictability (since they generally do not seek to provide a security-preserving refinement). Our security property rules this out. This should not be seen as a restriction: Nondeterministic choice playing a functional role can always be modelled by explicitely generating coins and branching on them.

Many secrecy properties follow one of the following two approaches (discussed in [RS99, Aba00]; an example for a different approach can be found in [Sch96]). One is based on equivalences: Suppose a process specification P is parameterised over a variable x representing a piece of data whose secrecy should be preserved. The idea is that P preserves the secrecy of this data if for any two data values d_0, d_1 substituted for x, the resulting processes $P(d_0)$ and $P(d_1)$ are equivalent, i. e. indistinguishable to any adversary, (this appears e. g. in [AG99]). This kind of secrecy property ensures a rather high degree of security. However, if it should be preserved by the usual refinement, it seems to require a rather fine-grained model: The equivalence may only relate those traces in the trace sets of $P(d_0)$ and $P(d_1)$ that originate from the same nondeterministic component of P, because otherwise dropping nondeterministic components during refinement may not preserve the equivalence. Such a model can be constructed (e. g. using ideas from [Jür00a]), but it seems to be necessarily relatively complicated.

The secrecy property considered in this paper relies on the idea that a process specification preserves the secrecy of a piece of data d if the process never sends out any information from which d could be derived, even in interaction with an adversary (this is attributed to [DY83] and used e. g. in [CGG00]; a similar notion is used in [SV00]). In general, it is slightly coarser than the first kind in that it may not prevent implicit information flow, but both kinds of security properties seem to be roughly equivalent in practice [Aba00]. But even a secrecy property that uncovers only *most* flaws but is preserved under standard refinements is useful enough, especially since more fine-grained security properties may be hard to ensure in practice, as pointed out in [RS99].

With a secrecy-preserving refinement, one can also address situations where implementations of formally verified security protocols turn out to be insecure, because the cryptographic primitive chosen in the implementation introduces new equalities between terms (as pointed out in [RS98]) by proving the nondeterministic sum of the protocol behaviour each with the different primitives, and thus deriving the security wrt. to each primitive separately.

Related Work For results on the use of formal methods in the development of secure systems cf. [FBGL94]. The survey article [Mea96] identifies the idea of security by design as a major area for future research.

In [Lot00], threat scenarios are used to formally develop secure systems using Focus. The considered security properties do not seem to be preserved under refinement and issues of refinement are left for further work.

[Sch96] gives a confidentiality property preserved under refinement. However, cryptographic primitives are not considered and it is pointed out that their treatment may be less straightforward.

For a discussion on refinement of secure information flow properties cf. [GCS91, Mea92, McL94, McL96]. [RWW94] avoids the "refinement paradox" by giving a security property that requires systems to appear deterministic to the untrusted environment. Special refinement operators that preserve information flow security are considered e. g. in [Man00].

A related problem is that formal specifications involving cryptographic operations usually assume unconditional security while implementations generally provide security only against adversaries with bounded resources. This problem is addressed in [AR00, AJ00] (the second article considers our model here).

2 Specification Language

In this work, we view specifications as nondeterministic programs in the specification framework Focus [BS00]. Note that in addition to these executable specifications, Focus also allows the use of non-executable specifications ((non-)executability of security specifications is discussed in [LFBG95]). Executable specifications allow a rather straightforward modelling of cryptographic aspects such as encryption.

Specifically, we consider concurrently executing processes interacting by transmitting sequences of data values over unidirectional FIFO communication channels. Communication is asynchronous in the sense that transmission of a value cannot be prevented by the receiver (note that one may model synchronous communication using handshake [BS00]).

Focus provides mechanical assistance in form of the CASE tool Autofocus [HMR$^+$98].

Processes are collections of programs that communicate synchronously (in rounds) through channels, with the constraint that for each of its output channels c the process contains exactly one program p_c that outputs on c. This program p_c may take input from any of P's input channels. Intuitively, the program is a description of a value to be output on the channel c in round $n+1$, computed from values found on channels in round n. Local state can be maintained through the use of feedback channels, and used for iteration (for instance, for coding *while* loops).

To be able to reason inductively on syntax, we use a simple specification language from [AJ00, Jür01]. We assume disjoint sets $\mathcal{D}$ of data values, **Secret** of unguessable values, **Keys** of keys, **Channels** of channels and **Var** of variables. Write $\mathbf{Enc} \stackrel{\text{def}}{=} \mathbf{Keys} \cup \mathbf{Channels} \cup \mathbf{Var}$ for the set of *encryptors* that may be used for encryption or decryption. The values communicated over channels are formal *expressions* built from variables, values on input channels, and data values using concatenation. Precisely, the set **Exp** of expressions contains the empty expression ε and the non-empty expressions generated by the grammar

$E ::=$	expression
d	data value ($d \in \mathcal{D}$)
N	unguessable value ($N \in \mathbf{Secret}$)
K	key ($K \in \mathbf{Keys}$)
c	input on channel c ($c \in \mathbf{Channels}$)
x	variable ($x \in \mathbf{Var}$)
$E_1 :: E_2$	concatenation
$\{E\}_e$	encryption ($e \in \mathbf{Enc}$)

$\mathcal{D}ec_e(E)$	decryption ($e \in \mathbf{Enc}$)

An occurrence of a channel name c refers to the value found on c at the previous instant. The empty expression ε denotes absence of output on a channel at a given point in time. We write **CExp** for the set of *closed* expressions (those containing no subterms in $\mathbf{Var} \cup \mathbf{Channels}$). We write the decryption key corresponding to an encryption key K as K^{-1}. In the case of asymmetric encryption, the encryption key K is public, and K^{-1} secret. For symmetric encryption, K and K^{-1} may coincide. We assume $\mathcal{D}ec_{K^{-1}}(\{E\}_K) = E$ for all $E \in \mathbf{Exp}, K, K^{-1} \in \mathbf{Keys}$ (and we assume that no other equations except those following from these hold, unless stated otherwise).

(Non-deterministic) programs are defined by the grammar:

$p ::=$	programs
E	output expression ($E \in \mathbf{Exp}$)
either p *or* p'	nondeterministic branching
if $E = E'$ *then* p *else* p'	conditional ($E, E' \in \mathbf{Exp}$)
case E *of key do* p *else* p'	determine if E is a key ($E \in \mathbf{Exp}$)
case E *of* $x :: y$ *do* p *else* p'	break up list into head::tail ($E \in \mathbf{Exp}$)

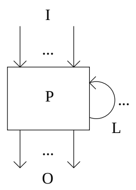
Variables are introduced in case constructs, which determine their values. The first case construct tests whether E is a key; if so, p is executed, otherwise p'. The second case construct tests whether E is a list with head x and tail y; if so, p is evaluated, using the actual values of x, y; if not, p' is evaluated. In the second case construct, x and y are bound variables. A program is *closed* if it contains no unbound variables. *while* loops can be coded using feedback channels.

From each assignment of expressions to channel names $c \in \mathbf{Channels}$ appearing in a program p (called its *input channels*), p computes an output expression.

For simplification we assume that in the following all programs are *well-formed* in the sense that each encryption $\{E\}_e$ and decryption $\mathcal{D}ec_e(E)$ appears as part of p in a *case* E' *of key do* p *else* p' construct (unless $e \in \mathbf{Keys}$), to ensure that only keys are used to encrypt or decrypt. It is straightforward to enforce this using a type system.

Example The program *case* c *of key do* $\{d\}_c$ *else* ε outputs the value received at channel d encrypted under the value received on channel c if that value is a key, otherwise it outputs ε.

A *process* is of the form $P = (I, O, L, (p_c)_{c \in O \cup L})$ where



- $I \subseteq$ **Channels** is called the set of its *input channels* and
- $O \subseteq$ **Channels** the set of its *output channels*,

and where for each $c \in \tilde{O} \stackrel{\text{def}}{=} O \cup L$, p_c is a closed program with input channels in $\tilde{I} \stackrel{\text{def}}{=} I \cup L$ (where $L \subseteq$ **Channels** is called the set of *local channels*). From inputs on the channels in $\tilde{I}$ at a given point in time, p_c computes the output on the channel c.

We write I_P, O_P and L_P for the sets of input, output and local channels of P, $K_P \subseteq$ **Keys** for the set of private keys and $S_P \subseteq$ **Secret** for the set of unguessable values (such as nonces) occurring in P. We assume that different processes have disjoint sets of local channels, keys and secrets. Local channels are used to store local state between the execution rounds.

2.1 Stream-processing functions

In this subsection we recall the definitions of streams and stream-processing functions from [Bro99, BS00].

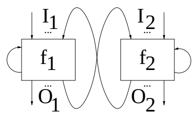
We write $\mathbf{Stream}_C \stackrel{\text{def}}{=} (\mathbf{CExp}^{\omega})^C$ (where $C \subseteq$ **Channels**) for the set of C-indexed tuples of (finite or infinite) sequences of closed expressions. The elements of this set are called *streams*, specifically *input streams* (resp. *output streams*) if C denotes the set of non-local input (resp. output) channels of a process P. Each stream $\boldsymbol{s} \in \mathbf{Stream}_C$ consists of components $\boldsymbol{s}(c)$ (for each $c \in C$) that denote the sequence of expressions appearing at the channel c. The n^{th} element in this sequence is the expression appearing at time $t = n$.

A function $f : \mathbf{Stream}_I \to \mathcal{P}(\mathbf{Stream}_O)$ from streams to sets of streams is called a *stream-processing function*.

The composition of two stream-processing functions $f_i : \mathbf{Stream}_{I_i} \to \mathcal{P}(\mathbf{Stream}_{O_i})$ $(i = 1, 2)$ with $O_1 \cap O_2 = \emptyset$ is defined as

$$f_1 \otimes f_2 : \mathbf{Stream}_I \to \mathcal{P}(\mathbf{Stream}_O)$$

(with $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$, $O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$).



where $f_1 \otimes f_2(\boldsymbol{s}) \stackrel{\text{def}}{=} \{\boldsymbol{t}\restriction_O : \boldsymbol{t}\restriction_I = \boldsymbol{s}\restriction_I \wedge \boldsymbol{t}\restriction_{O_i} \in f_i(\boldsymbol{s}\restriction_{I_i})(i = 1, 2)\}$ (where $\boldsymbol{t}$ ranges over $\mathbf{Stream}_{I \cup O}$). For $\boldsymbol{t} \in \mathbf{Stream}_C$ and $C' \subseteq C$, the restriction $\boldsymbol{t}\restriction_{C'} \in \mathbf{Stream}_{C'}$ is defined by $\boldsymbol{t}\restriction_{C'}(c) = \boldsymbol{t}(c)$ for each $c \in C'$.

$$
\begin{array}{ll}
[E](\boldsymbol{M}) = \{E(\boldsymbol{M})\} & \text{where } E \in \mathbf{Exp} \\
[\mathit{either}\ p\ \mathit{or}\ p'](\boldsymbol{M}) = [p](\boldsymbol{M}) \cup [p'](\boldsymbol{M}) & \\
[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'](\boldsymbol{M}) = [p](\boldsymbol{M}) & \text{if } [E](\boldsymbol{M}) = [E'](\boldsymbol{M}) \\
[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'](\boldsymbol{M}) = [p'](\boldsymbol{M}) & \text{if } [E](\boldsymbol{M}) \neq [E'](\boldsymbol{M}) \\
[\mathit{case}\ E\ \mathit{of}\ \mathit{key}\ \mathit{do}\ p\ \mathit{else}\ p'](\boldsymbol{M}) = [p](\boldsymbol{M}) & \text{if } [E](\boldsymbol{M}) \in \mathbf{Keys} \\
[\mathit{case}\ E\ \mathit{of}\ \mathit{key}\ \mathit{do}\ p\ \mathit{else}\ p'](\boldsymbol{M}) = [p'](\boldsymbol{M}) & \text{if } [E](\boldsymbol{M}) \notin \mathbf{Keys} \\
[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'](\boldsymbol{M}) = [p[h/x, t/y]](\boldsymbol{M}) & \text{if } [E](\boldsymbol{M}) = h :: t \\
\qquad\qquad \text{where } h \neq \varepsilon \text{ and } h \text{ is not of the form } & h_1 :: h_2 \text{ for } h_1, h_2 \neq \varepsilon \\
[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'](\boldsymbol{M}) = [p'](\boldsymbol{M}) & \text{if } [E](\boldsymbol{M}) = \varepsilon
\end{array}
$$

Fig. 1. Definition of $[p](\boldsymbol{M})$.

Since the operator $\otimes$ is associative and commutative [BS00], we can define a generalised composition operator $\bigotimes_{i \in I} f_i$ for a set $\{f_i : i \in I\}$ of stream-processing functions.

Example If $f : \mathbf{Stream}_{\{a\}} \to \mathcal{P}(\mathbf{Stream}_{\{b\}})$, $f(\boldsymbol{s}) \stackrel{\text{def}}{=} \{0.\boldsymbol{s}, 1.\boldsymbol{s}\}$, is the stream-processing function with input channel a and output channel b that outputs the input stream prefixed with either 0 or 1, and $g : \mathbf{Stream}_{\{b\}} \to \mathcal{P}(\mathbf{Stream}_{\{c\}})$, $g(\boldsymbol{s}) \stackrel{\text{def}}{=} \{0.\boldsymbol{s}, 1.\boldsymbol{s}\}$, the function with input (resp. output) channel b (resp. c) that does the same, then the composition $f \otimes g : \mathbf{Stream}_{\{a\}} \to \mathcal{P}(\mathbf{Stream}_{\{c\}})$, $f \otimes g(\boldsymbol{s}) = \{0.0.\boldsymbol{s}, 0.1.\boldsymbol{s}, 1.0.\boldsymbol{s}, 1.1.\boldsymbol{s}\}$, outputs the input stream prefixed with either of the 2-element streams 0.0, 0.1, 1.0 or 1.1.

2.2 Associating a stream-processing function to a process

A process $P = (I, O, L, (p_c)_{c \in O})$ is modelled by a stream-processing function $[\![P]\!] : \mathbf{Stream}_I \to \mathcal{P}(\mathbf{Stream}_O)$ from input streams to sets of output streams.

For honest processes P, $[\![P]\!]$ is by construction *causal*, which means that the $n + 1^{st}$ expression in any output sequence depends only on the first n input expressions. As pointed out in [Pfi98], adversaries can not be assumed to behave causally, therefore for an adversary A we need a slightly different interpretation $[\![A]\!]_r$ (called *sometimes rushing adversaries* in [Pfi98]).

For any closed program p with input channels in $\tilde{I}$ and any $\tilde{I}$-indexed tuple of closed expressions $\boldsymbol{M} \in \mathbf{CExp}^{\tilde{I}}$ we define a set of expressions $[p](\boldsymbol{M}) \in \mathcal{P}(\mathbf{CExp})$ in Figure 1, so that $[p](\boldsymbol{M})$ is the expression that results from running p once, when the channels have the initial values given in $\boldsymbol{M}$.

We write $E(\boldsymbol{M})$ for the result of substituting each occurrence of $c \in \tilde{I}$ in E by $\boldsymbol{M}(c)$ and $p[E/x]$ for the outcome of replacing each free occurrence of x in process P with the term E, renaming variables to avoid capture.

Then any program p_c (for $c \in \mathbf{Channels}$) defines a causal stream-processing function $[p_c] : \mathbf{Stream}_{\tilde{I}} \to \mathcal{P}(\mathbf{Stream}_{\{c\}})$ as follows. Given $\boldsymbol{s} \in \mathbf{Stream}_{\tilde{I}}$, let $[p_c](\boldsymbol{s})$ consist of those $\boldsymbol{t} \in \mathbf{Stream}_{\{c\}}$ such that

- $\boldsymbol{t}_0 \in [p_c](\varepsilon, \ldots, \varepsilon)$
- $\boldsymbol{t}_{n+1} \in [p_c](\boldsymbol{s}_n)$ for each $n \in \mathbb{N}$.

Finally, a process $P = (I, O, L, (p_c)_{c \in \tilde{O}})$ is interpreted as the composition $[\![P]\!] \stackrel{\text{def}}{=} \bigotimes_{c \in \tilde{O}} [p_c]$.

Similarly, any p_c (with $c \in \mathbf{Channels}$) defines a non-causal stream-processing function $[p_c]_r : \mathbf{Stream}_{\tilde{I}} \to \mathcal{P}(\mathbf{Stream}_{\{c\}})$ as follows. Given $\boldsymbol{s} \in \mathbf{Stream}_{\tilde{I}}$, let $[p_c]_r(\boldsymbol{s})$ consist of those $\boldsymbol{t} \in \mathbf{Stream}_{\{c\}}$ such that $\boldsymbol{t}_n \in [p_c]_r(\boldsymbol{s}_n)$ for each $n \in \mathbb{N}$.

An adversary $A = (I, O, L, (p_c)_{c \in \tilde{O}})$ is interpreted as the composition $[\![A]\!]_r \stackrel{\text{def}}{=} \bigotimes_{c \in O} [p_c]_r \otimes \bigotimes_{l \in \tilde{O} \setminus O} [p_l]$. Thus the programs with outputs on the non-local channels are defined to be *rushing* (note that using the local channels an adversary can still show causal behaviour).

Examples

- $[\mathit{if}\ \mathcal{D}ec_{K'}(\{0\}_K) = 0\ \mathit{then}\ 0\ \mathit{else}\ 1](\boldsymbol{s}) = (0, 0, 0, \ldots)$ iff $K = K'$
- For the process P with $I_P = \{i\}$, $O_P = \{o\}$ and $L_P = \{l\}$ and with $p_l \stackrel{\text{def}}{=} l :: i$ and $p_o \stackrel{\text{def}}{=} l :: i$ we have $[\![P]\!](\boldsymbol{s}) = \{(\varepsilon, \boldsymbol{s}_0, \boldsymbol{s}_0 :: \boldsymbol{s}_1, \boldsymbol{s}_0 :: \boldsymbol{s}_1 :: \boldsymbol{s}_2, \ldots)\}$ and $[\![P]\!]_r(\boldsymbol{s}) = \{(\boldsymbol{s}_0, \boldsymbol{s}_0 :: \boldsymbol{s}_1, \boldsymbol{s}_0 :: \boldsymbol{s}_1 :: \boldsymbol{s}_2, \ldots)\}$.

3 Secrecy

We say that a stream-processing function $f : \mathbf{Stream}_{\emptyset} \to \mathcal{P}(\mathbf{Stream}_O)$ *may eventually output* an expression $E \in \mathbf{CExp}$ if there exists a stream $\boldsymbol{t} \in f(*)$ (where $*$ denotes the sole element in $\mathbf{Stream}_{\emptyset}$), a channel $c \in O$ and an index $j \in \mathbb{N}$ such that $(\boldsymbol{t}(c))_j = E$.

Definition 1. *We say that a process P* leaks *a secret $m \in \mathbf{Secret} \cup \mathbf{Keys}$ if there is a process A with $I_A \subseteq O_P$, $I_P \subseteq O_A$ and $m \notin S_A \cup K_A$ such that $[\![P]\!] \otimes [\![A]\!]_r$ may eventually output m. Otherwise we say that P* preserves *the secrecy of m.*

The idea of this definition is that P preserves the secrecy of m if no adversary can find out m in interaction with P. In our formulation m is either an unguessable value or a key; this is seen to be sufficient in practice, since the secrecy of a compound expression can usually be derived from that of a key or unguessable value [Aba00].

For a comparison with other secrecy properties cf. Section 1.

Examples

- $p \stackrel{\text{def}}{=} \{m\}_K :: K$ does not preserve the secrecy of m or K, but $p \stackrel{\text{def}}{=} \{m\}_K$ does.
- $p_l \stackrel{\text{def}}{=}$ *case* c *of key do* $\{m\}_c$ *else* ε (where $c \in \mathbf{Channels}$) does not preserve the secrecy of m, but $P \stackrel{\text{def}}{=} (\{c\}, \{e\}, \{l\}, (p_l, p_e))$ (where $p_e \stackrel{\text{def}}{=} \{l\}_K$) does.

We also define a rely-guarantee condition for secrecy.

Given a relation $C \subseteq \mathbf{Stream}_O \times \mathbf{Stream}_I$ and a process A with $O \subseteq I_A$ and $I \subseteq O_A$ we say that A *fulfils* C if for every $\boldsymbol{s} \in \mathbf{Stream}_{I_A}$ and every $\boldsymbol{t} \in [\![A]\!](\boldsymbol{s})$, we have $(\boldsymbol{s}\upharpoonright_O, \boldsymbol{t}\upharpoonright_I) \in C$.

Definition 2. *Given a relation $C \subseteq \mathbf{Stream}_{O_P} \times \mathbf{Stream}_{I_P}$ from output streams of a process P to input streams of P, we say that P* leaks m assuming C *(for $m \in \mathbf{Secret} \cup \mathbf{Keys}$) if there exists a process A with $m \notin S_A \cup K_A$ that fulfils C and such that $[\![P]\!] \otimes [\![A]\!]_r$ may eventually output m.*
Otherwise P preserves the secrecy of m assuming C.

This definition is useful if P is a component of a larger system S that is assumed to fulfil the rely-condition, or if the adversary is assumed to be unable to violate it.

Example $p \stackrel{\text{def}}{=}$ *if* $c = \mathbf{password}$ *then* $\mathbf{secret}$ *else* ε preserves the secrecy of *secret* assuming $C = \{(\boldsymbol{t}, \boldsymbol{s}) : \forall n. \boldsymbol{s}_n \neq \mathbf{password}\}$.

4 Refinement

We define various notions of refinement given in [BS00] and exhibit conditions under which they preserve our proposed secrecy properties.

4.1 Property refinement

Definition 3. *For processes P and P' with $I_P = I_{P'}$ and $O_P = O_{P'}$ we define $P \rightsquigarrow P'$ if for each $\boldsymbol{s} \in \mathbf{Stream}_{I_P}$, $[\![P]\!](\boldsymbol{s}) \supseteq [\![P']\!](\boldsymbol{s})$.*

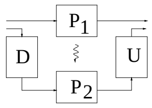
Example ($either\ p\ or\ q$) $\rightsquigarrow p$ and ($either\ p\ or\ q$) $\rightsquigarrow q$ for any programs p, q.

Theorem 1.

- *If P preserves the secrecy of m and $P \rightsquigarrow P'$ then P' preserves the secrecy of m.*
- *If P preserves the secrecy of m assuming C (for any $C \subseteq \mathbf{Stream}_{O_P} \times \mathbf{Stream}_{I_P}$) and $P \rightsquigarrow P'$ then P' preserves the secrecy of m assuming C.*

4.2 Interface refinement

Definition 4. *Let P_1, P_2, D and U be processes with $I_{P_1} = I_D$, $O_D = I_{P_2}$, $O_{P_2} = I_U$ and $O_U = O_{P_1}$. We define $P_1 \overset{(D,U)}{\rightsquigarrow} P_2$ to hold if $P_1 \rightsquigarrow D \otimes P_2 \otimes U$.*



Example Suppose we have

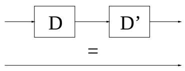
- $P_1 = (\{c\}, \{d\}, p_d \overset{\text{def}}{=} if\ c = 1\ then\ 2\ else\ 3)$,
- $P_2 = (\{c'\}, \{d'\}, p_{d'} \overset{\text{def}}{=} if\ c' = 4\ then\ 5\ else\ 6)$,
- $D = (\{c\}, \{c'\}, p_{c'} \overset{\text{def}}{=} if\ c = 1\ then\ 4\ else\ \varepsilon)$ and
- $U = (\{d'\}, \{d\}, p_d \overset{\text{def}}{=} if\ d' = 5\ then\ 2\ else\ 3$.

Then we have $P_1 \overset{(D,U)}{\rightsquigarrow} P_2$.

For the preservation result we need the following concepts.

Given a stream $\boldsymbol{s} \in \mathbf{Stream}_X$ and a bijection $\iota : Y \to X$ we write $\boldsymbol{s}_\iota$ for the stream in $\mathbf{Stream}_Y$ obtained from $\boldsymbol{s}$ by renaming the channel names using ι: $\boldsymbol{s}_\iota(y) = \boldsymbol{s}(\iota(y))$.

Given processes D, D' with $O_D = I_{D'}$ and $O_{D'} \cap I_D = \emptyset$ and a bijection $\iota : O_{D'} \to I_D$ such that $[\![D]\!] \otimes [\![D']\!](\boldsymbol{s}) = \{\boldsymbol{s}_\iota\}$ for each $\boldsymbol{s} \in \mathbf{Stream}_{I_D}$, we say that D is a *left inverse* of D' and D' is a *right inverse* of D.



Example $p_d \overset{\text{def}}{=} 0 :: c$ is a left inverse of $p_e \overset{\text{def}}{=} case\ c\ of\ h :: t\ do\ t\ else\ \varepsilon$.

We write $S \circ R \overset{\text{def}}{=} \{(x, z) : \exists y.(x, y) \in R \wedge (y, z) \in S\}$ for the usual composition of relations R, S and generalize this to functions $f : X \to \mathcal{P}(Y)$ by viewing them as relations $f \subseteq X \times Y$.

Theorem 1 *Let P_1, P_2, D and U be processes with $I_{P_1} = I_D$, $O_D = I_{P_2}$, $O_{P_2} = I_U$ and $O_U = O_{P_1}$ and such that D has a left inverse D' and U a right inverse U'. Let $m \in (\mathbf{Secret} \cup \mathbf{Keys}) \setminus \bigcup_{Q \in \{D', U'\}} (S_Q \cup K_Q)$.*

– *If* P_1 *preserves the secrecy of* m *and* $P_1 \overset{(D,U)}{\rightsquigarrow} P_2$ *then* P_2 *preserves the secrecy of* m.
– *If* P_1 *preserves the secrecy of* m *assuming* $C \subseteq \mathbf{Stream}_{O_{P_1}} \times \mathbf{Stream}_{I_{P_1}}$ *and* $P_1 \overset{(D,U)}{\rightsquigarrow} P_2$ *then* P_2 *preserves the secrecy of* m *assuming* $[\![U']\!] \circ C \circ [\![D']\!]$.

4.3 Conditional refinement

Definition 5. *Let* P_1 *and* P_2 *be processes with* $I_{P_1} = I_{P_2}$ *and* $O_{P_1} = O_{P_2}$. *We define* $P_1 \rightsquigarrow_C P_2$ *for a total relation* $C \subseteq \mathbf{Stream}_{O_{P_1}} \times \mathbf{Stream}_{I_{P_1}}$ *to hold if for each* $\boldsymbol{s} \in \mathbf{Stream}_{I_{P_1}}$ *and each* $\boldsymbol{t} \in [\![P_2]\!]$, $(\boldsymbol{t}, \boldsymbol{s}) \in C$ *implies* $\boldsymbol{t} \in [\![P_1]\!]$.

Example $p \rightsquigarrow_C$ (*if* $c =$ **emergency** *then* q *else* p) for $C = \{(\boldsymbol{t}, \boldsymbol{s}) : \forall n. \boldsymbol{s}_n \neq \mathbf{emergency}\}$.

Theorem 2
Given total relations $C, D \subseteq \mathbf{Stream}_{O_P} \times \mathbf{Stream}_{I_P}$ *with* $C \subseteq D$, *if* P *preserves the secrecy of* m *assuming* C *and* $P \rightsquigarrow_D P'$ *then* P' *preserves the secrecy of* m *assuming* C.

5 A variant of TLS

To demonstrate usability of our specification framework we specify a variant of the handshake protocol of TLS as proposed in [APS99] and demonstrate a previously unpublished weakness.

5.1 The Handshake Protocol

The goal is to let a client C send a master secret $m \in \mathbf{Secret}$ to a server S in a way that provides confidentiality and server authentication.

The protocol uses both RSA encryption and signing. Thus in this and the following section we assume also the equation $\{\mathcal{D}ec_{K^{-1}}(E)\}_K = E$ to hold (for each $E \in \mathbf{Exp}$ and $K \in \mathbf{Keys}$). We also assume that the set of data values $\mathcal{D}$ includes process names such as $C, S, Y, \ldots$ and a message *abort*.

The protocol assumes that there is a secure (wrt. integrity) way for C to obtain the public key K_{CA} of the certification authority, and for S to obtain a certificate $\mathcal{D}ec_{K_{CA}^{-1}}(S :: K_S)$ signed by the certification authority that contains its name and public key. The adversary may also have access to K_{CA}, $\mathcal{D}ec_{K_{CA}^{-1}}(S :: K_S)$ and $\mathcal{D}ec_{K_{CA}^{-1}}(Z :: K_Z)$ for an arbitrary process Z.

The channels between the participants are thus as follows.

$$\begin{tikzcd}
C \arrow[loop left, "l"] \arrow[r, bend left, "c"] & A \arrow[l, bend left, "s'"] \arrow[r, bend left, "c'"] & S \arrow[l, bend left, "s"] \\
& CA \arrow[ul, "a_C"] \arrow[u, "a_A"] \arrow[ur, "a_S"'] &
\end{tikzcd}$$

The following is the message flow diagram for the protocol that we present to aid understanding. Note that this kind of notation is merely short-hand for the more explicit specification given below and needs to be interpreted with care [Aba00].

$$C \xrightarrow{\quad N_C::K_C::\mathcal{D}ec_{K_C^{-1}}(C::K_C) \quad} S$$

$$C \xleftarrow{\quad N_S::\{\mathcal{D}ec_{K_S^{-1}}(K_{CS}::N_C)\}_{K_C}::\mathcal{D}ec_{K_{CA}^{-1}}(S::K_S) \quad} S$$

$$C \xrightarrow{\quad \{m\}_{K_{CS}} \quad} S$$

Now we specify the protocol in our specification framework (here and in the following we denote a program with output channel c simply as c for readability).

$$\begin{array}{l}
c \stackrel{\text{def}}{=} \mathit{if}\ l = \varepsilon\ \mathit{then}\ N_C :: K_C :: \mathcal{D}ec_{K_C^{-1}}(C :: K_C) \\
\qquad\qquad\quad \mathit{else}\ \mathit{case}\ s'\ \mathit{of}\ s_1 :: s_2 :: s_3 \\
\qquad\qquad\qquad\quad \mathit{do}\ \mathit{case}\ \{s_3\}_{a_C}\ \mathit{of}\ S :: x \\
\qquad\qquad\qquad\qquad\qquad \mathit{do}\ \mathit{if}\ \{\mathcal{D}ec_{K_C^{-1}}(s_2)\}_x = y :: N_C\ \mathit{then}\ \{m\}_y \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{else}\ \text{abort} \\
\qquad\qquad\qquad\qquad\qquad \mathit{else}\ \text{abort} \\
\qquad\qquad\qquad\quad \mathit{else}\ \varepsilon \\
l \stackrel{\text{def}}{=} 0 \\
s \stackrel{\text{def}}{=} \mathit{case}\ c'\ \mathit{of}\ c_1 :: c_2 :: c_3 \\
\qquad\qquad \mathit{do}\ \ \mathit{case}\ \{c_3\}_{c_2} \mathit{of}\ x :: c_2\ \mathit{do}\ N_S :: \{\mathcal{D}ec_{K_S^{-1}}(K_{CS} :: c_1)\}_{c_2} :: a_S \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{else}\ \text{abort} \\
\qquad\qquad \mathit{else}\ \varepsilon
\end{array}$$

$$a_C \stackrel{\text{def}}{=} K_{CA}$$
$$a_A \stackrel{\text{def}}{=} K_{CA} :: \mathcal{D}ec_{K_{CA}^{-1}}(S :: K_S) :: \mathcal{D}ec_{K_{CA}^{-1}}(Z :: K_Z)$$
$$a_S \stackrel{\text{def}}{=} \mathcal{D}ec_{K_{CA}^{-1}}(S :: K_S)$$

For readability we leave out a time-stamp, a session id, the choice of cipher suite and compression method and the use of a temporary key by S since these are not relevant for the weakness. We use syntactic sugar by extending the case list construct to lists of finite length and by using pattern matching, and we also leave out some *case of key do else* constructs to avoid cluttering. Here the local channel l of C only ensures that C initiates the handshake protocol only once. The exchanged key is symmetric, i. e. we have $K_{CS}^{-1} = K_{CS}$. The values sent on a_A signify that we allow A to eavesdrop on a_C and a_S and to obtain the certificate issued by CA of some third party.

5.2 The flaw

Theorem 3 *$P \stackrel{\text{def}}{=} C \otimes S \otimes CA$ does not preserve the secrecy of m.*

We specify an attacker A and show that it is successful.

$$\begin{aligned}
c' &\stackrel{\text{def}}{=} \mathit{case}\ c\ \mathit{of}\ c_1 :: c_2 :: c_3 \\
&\qquad\qquad \mathit{do}\ c_1 :: K_A :: \mathcal{D}ec_{K_A^{-1}}(C :: K_A) \\
&\qquad\qquad \mathit{else}\ \varepsilon \\
s' &\stackrel{\text{def}}{=} \mathit{case}\ s\ \mathit{of}\ s_1 :: s_2 :: s_3 \\
&\qquad\qquad \mathit{do}\ s_1 :: \{\mathcal{D}ec_{K_A^{-1}}(s_2)\}_{K_C} :: s_3 \\
&\qquad\qquad \mathit{else}\ \varepsilon \\
l_A &\stackrel{\text{def}}{=} \mathit{if}\ l_A = \varepsilon\ \mathit{then}\ \mathit{case}\ s\ \mathit{of}\ s_1 :: s_2 :: s_3 \\
&\qquad\qquad\qquad \mathit{do}\ \mathit{case}\ \{\mathcal{D}ec_{K_A^{-1}}(s_2)\}_{K_S}\ \mathit{of}\ x_1 :: x_2\ \mathit{do}\ x_1\ \mathit{else}\ l_A \\
&\qquad\qquad\quad \mathit{else}\ l_A \\
c_0 &\stackrel{\text{def}}{=} \mathit{case}\ l_A\ \mathit{of}\ \mathit{key}\ \mathit{do}\ \mathit{if}\ \mathcal{D}ec_{l_A}(c) = \perp\ \mathit{then}\ \varepsilon\ \mathit{else}\ \mathcal{D}ec_{l_A}(c)\ \mathit{else}\ \varepsilon
\end{aligned}$$

Proposition 1 *$[\![P]\!] \otimes [\![A]\!]_r$ eventually outputs m.*

The message flow diagram corresponding to this man-in-the-middle attack follows.

$$C \xrightarrow{N_C::K_C::\mathcal{D}ec_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_C::K_A::\mathcal{D}ec_{K_A^{-1}}(C::K_A)} S$$

$$C \xleftarrow{N_S::\{\mathcal{D}ec_{K_S^{-1}}(K_{CS}::N_C)\}_{K_C}::\mathcal{D}ec_{K_{CA}^{-1}}(S::K_S)} A \xleftarrow{N_S::\{\mathcal{D}ec_{K_S^{-1}}(K_{CS}::N_C)\}_{K_A}::\mathcal{D}ec_{K_{CA}^{-1}}(S::K_S)} S$$

$$C \xrightarrow{\{m\}_{K_{CS}}} A \xrightarrow{\{m\}_{K_{CS}}} S$$

5.3 The fix

Let S' be the process derived from S by substituting $K_{CS} :: c_1$ in the second line of the definition of s by $K_{CS} :: c_1 :: c_2$. Change C to C' by substituting $y :: N_C$ in the fourth line of the definition of c by $y :: N_C :: K_C$.

$$C \xrightarrow{N_C::K_C::\mathcal{D}ec_{K_C^{-1}}(C::K_C)} S$$

$$C \xleftarrow{N_S::\{\mathcal{D}ec_{K_S^{-1}}(K_{CS}::N_C::\underline{K_C})\}_{K_C}::\mathcal{D}ec_{K_{CA}^{-1}}(S::K_S)} S$$

$$C \xrightarrow{\{m\}_{K_{CS}}} S$$

Theorem 4 *$P' \stackrel{\text{def}}{=} C' \otimes S' \otimes CA$ preserves the secrecy of m.*

Proof. For lack of space we only give an informal (but mathematically precise) sketch of the proof.

Given an adversary A with $n \notin S_A \cup K_A$, we need to show that $[\![P']\!] \otimes [\![A]\!]_r$ does not eventually output m. We proceed by execution rounds, making use of the fact that the adversary may let its output depend on the output from the honest participants at the same time.

In every round, 0 is output on l, K_{CA} on a_C and a_A, and $\mathcal{D}ec_{K_{CA}^{-1}}(S :: K_S)$ on a_S. After the first round, the local storage of C remains unchanged whatever happens, and S and CA do not have a local storage. Thus we only need to consider those actions of A that immediately increase its knowledge (i. e. we need not consider outputs of A that prompt C or S to output ε or *abort* in the following round.

In the first round, $N_C :: K_C :: \mathcal{D}ec_{K_C^{-1}}(C :: K_C)$ is output on c and ε on s. Since A is not in possession of any message containing S's name and signed by CA at this point, any output on s' will prompt C to output ε or *abort* in the next round, so the output on s' is irrelevant. Similarly, the only relevant output on c' is of the form $c_1 :: K_X :: \mathcal{D}ec_{K_X^{-1}}(Y :: K_X)$, where K_X is a public key with corresponding private key K_X^{-1} and Y a name of a process.

In the second round, the output on c is ε or *abort*, and that on s is ε or *abort* or $N_S :: \{\mathcal{D}ec_{K_S^{-1}}(K_{CS} :: c_1 :: K_X)\}_{K_X} :: a_S$. The only possibility to cause C in the following round to produce a relevant output would be for A now to output a message of the form $N_Z :: \{\mathcal{D}ec_{K_Z^{-1}}(K_{CS} :: c_1 :: K_X)\}_{K_X} :: \mathcal{D}ec_{K_{CA}^{-1}}(S :: K_Z)$. Firstly, the only certificate from CA containing S in possession of A is $\mathcal{D}ec_{K_{CA}^{-1}}(S :: K_S)$. Secondly, the only message containing a message signed using K_S in possession of A is $\{\mathcal{D}ec_{K_S^{-1}}(K_{CS} :: c_1 :: K_X)\}_{K_X}$. In case $K_X \neq K_C$ the message signed by S is of the form $\mathcal{D}ec_{K_S^{-1}}(K_{CS} :: c_1 :: K_X)$ for $K_X \neq K_C$, so that C outputs *abort* on receipt of this message anyhow. In case $K_X = K_C$, A cannot decrypt or alter the message $\{\mathcal{D}ec_{K_S^{-1}}(K_{CS} :: c_1 :: K_C)\}_{K_C}$ by assumptions on cryptography and since A does not possess K_C^{-1}. A may forward the message on s'. In this case, C outputs $\{m\}_{K_{CS}}$ in the following round, which A cannot decrypt.

Since the internal state of C, S and CA does not change after the first round, further interaction does not bring any change whatsoever (since it makes no difference if A successively tries different keys K_X or names Y).

Thus P' preserves the secrecy of x.

Note that the nonce N_S is in fact superfluous.

6 Implementing Secure Channels

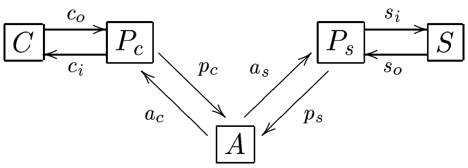
As an example for a stepwise development of a secure system from an abstract specification to a concrete one, we consider the implementation of a secure channel W from a client C to a server S using the handshake protocol considered in Section 5.

The initial requirement is that a client C should be able to send a message n on W with intended destination a server S so that n is not leaked to A. Before a security risk analysis the situation may simply be pictured as follows:

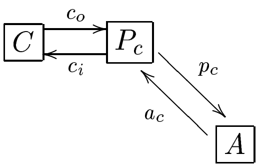
$$\begin{tikzcd} \boxed{C} \arrow[r, shift left, "c_o"] & \boxed{W} \arrow[l, shift left, "c_i"] \arrow[r, shift left, "s_i"] & \boxed{S} \arrow[l, shift left, "s_o"] \end{tikzcd}$$

Since there are no unconnected output channels, the composition $C \otimes W \otimes S$ obviously does not leak m.

Suppose that the risk analysis indicates that the transport layer over which W is to be implemented is vulnerable against active attacks. This leads to the following model.



We would like to implement the secure channel using the (corrected) variant of the TLS handshake protocol considered in Section 5. Thus P_c resp. P_s are implemented by making use of the client resp. server side of the handshake protocol. Here we only consider the client side:



We would like to provide an implementation P_c such that for each C with $n \in S_C$, $C \otimes P_c$ preserves the secrecy of n (where n represents the message that should be sent to S). Of course, P_c should also provide functionality: perform the initial handshake and then encrypt data from C under the negotiated key $K \in \mathbf{Keys}$ and sent it out onto the network. As a first step, we may formulate the possible outputs of P_c as nondeterministic choices (in order to constrain the overall behaviour of P_c). We also allow the possibility for P_c to signal to C the readiness to receive data to be sent over the network, by sending ok on c_i.

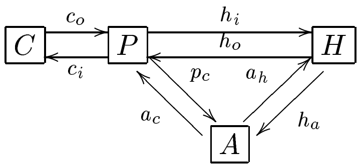
$$
\begin{aligned}
p_c &\stackrel{\text{def}}{=} \textit{either if } c_o = \varepsilon \textit{ then } \varepsilon \textit{ else } \{c_o\}_K \\
&\quad\ \textit{or } c_K \\
c_i &\stackrel{\text{def}}{=} \textit{either } \varepsilon \textit{ or } \text{ok}
\end{aligned}
$$

Here c_K denotes the following adaption of the (corrected) program c defined in Section 5 (for readability, we allow to use syntactic "macros" here, the resulting program is obtained by "pasting" the following program text in the place of $c(K)$ in the definition of p_c). For simplicity, we assume that P_c has already received the public key K_{CA} of the certification authority. We leave out the definition of c_i since at the moment we only consider the case where C wants to sent data to S.

$$
\begin{aligned}
c_K \stackrel{\mathrm{def}}{=} &\ \mathit{either}\ N_C :: K_C :: \mathcal{D}ec_{K_C^{-1}}(C :: K_C) \\
&\quad \mathit{or\ case}\ a_c\ \mathit{of}\ s_1 :: s_2 :: s_3 \\
&\qquad \mathit{do\ case}\ \mathcal{D}ec_{K_{AC}}(s_3)\ \mathit{of}\ S :: x \\
&\qquad\qquad \mathit{do\ if}\ \{\mathcal{D}ec_{K_C}(s_2)\}_x = y :: N_C :: K_C\ \mathit{then}\ \{K\}_y \\
&\qquad\qquad\qquad \mathit{else}\ \text{abort} \\
&\qquad\qquad \mathit{else}\ \text{abort} \\
&\qquad \mathit{else}\ \text{abort}
\end{aligned}
$$

One can show that for any C, the composition $C \otimes P_c$ preserves the secrecy of n.

As a next step, we may split P_c into two components: the client side H of the handshake protocol (as part of the security layer) and program P (in the application layer) that receives data from C, encrypts it using the key received from H and sends it out on the network:



$$
\begin{aligned}
h_a \stackrel{\mathrm{def}}{=} &\ \mathit{if}\ h_i = \varepsilon\ \mathit{then}\ N_C :: K_C :: \mathcal{D}ec_{K_C^{-1}}(C :: K_C) \\
&\quad \mathit{else\ case}\ a_h\ \mathit{of}\ s_1 :: s_2 :: s_3 \\
&\qquad \mathit{do\ case}\ \mathcal{D}ec_{K_{AC}}(s_3)\ \mathit{of}\ S :: x \\
&\qquad\qquad \mathit{do\ if}\ \{\mathcal{D}ec_{K_C}(s_2)\}_x = y :: N_C :: K_C\ \mathit{then}\ \{m\}_y \\
&\qquad\qquad\qquad \mathit{else}\ \text{abort} \\
&\qquad\qquad \mathit{else}\ \text{abort} \\
&\qquad \mathit{else}\ \text{abort} \\
h_o \stackrel{\mathrm{def}}{=} &\ \mathit{if}\ h_i = \varepsilon\ \mathit{then}\ \varepsilon \\
&\quad \mathit{else\ case}\ a_h\ \mathit{of}\ s_1 :: s_2 :: s_3 \\
&\qquad \mathit{do\ case}\ \mathcal{D}ec_{K_{AC}}(s_3)\ \mathit{of}\ S :: x \\
&\qquad\qquad \mathit{do\ if}\ \{\mathcal{D}ec_{K_C}(s_2)\}_x = y :: N_C :: K_C\ \mathit{then}\ \text{finished}
\end{aligned}
$$

$else\ \varepsilon$

$else\ \varepsilon$

$else\ \varepsilon$

$h_i \stackrel{\text{def}}{=} 0$

$p_c \stackrel{\text{def}}{=} \mathit{if}\ c_o = \varepsilon\ \mathit{then}\ \varepsilon\ \mathit{else}\ \{c_o\}_K$

$c_i \stackrel{\text{def}}{=} \mathit{if}\ h_o = \text{finished}\ \mathit{then}\ \text{ok}\ \mathit{else}\ \varepsilon$

We have the conditional interface refinement $P_c \overset{(D,U)}{\rightsquigarrow}_T P \otimes H$ where

- $T \subseteq \mathbf{Stream}_{O_{P_c}} \times \mathbf{Stream}_{I_{P_c}}$ consists of those $(\boldsymbol{s}, \boldsymbol{t})$ such that for any n, if $(\boldsymbol{s}(\tilde{c}_i)) \downharpoonright_i \neq \mathit{finished}$ for all $i \leq n$ then $(\boldsymbol{s}(\tilde{c}_o)) \downharpoonright_i = \varepsilon$ for all $i \leq n+1$
- and D and U have channel sets $I_D = \{\tilde{c}_o, \tilde{a}_c\}$, $O_D = \{c_o, a_c, a_h\}$, $I_U = \{c_i, p_c, h_a\}$ and $O_U = \{\tilde{c}_i, \tilde{p}_c\}$ and are specified by

$$c_o \stackrel{\text{def}}{=} \tilde{c}_o, \qquad a_c \stackrel{\text{def}}{=} \tilde{a}_c, \qquad a_h \stackrel{\text{def}}{=} \tilde{a}_c,$$
$$\tilde{c}_i \stackrel{\text{def}}{=} c_i, \qquad \tilde{p}_c \stackrel{\text{def}}{=} h_a$$

(after renaming the channels of P_c to $\tilde{c}_o, \tilde{c}_i, \tilde{p}_c, \tilde{a}_c$).

Therefore, for any C with $[\![C]\!] \subseteq T$, we have an interface refinement $C \otimes P_c \overset{(D,U)}{\rightsquigarrow} C \otimes P \otimes H$. Since for any C, the composition $C \otimes P_c$ preserves the secrecy of n, as noted above, this implies that for any C with $[\![C]\!] \subseteq T$, the composition $C \otimes P \otimes H$ preserves the secrecy of n by Theorem 1 (since D and U clearly have inverses).

7 Conclusion and Further Work

We presented work towards a framework for stepwise development of secure systems by showing a notion of secrecy (that follows a standard approach) to be preserved by standard refinement operators in the specification framework Focus. We gave a rely/guarantee version of the secrecy property and showed preservation by refinement. We used the secrecy property to uncover a previously unpublished flaw in a proposed variant of TLS, proposed a correction and proved it secure. We gave an abstract specification of a secure channel satisfying secrecy and refined it to a concrete specification, thus satisfying secrecy by the preservation result.

In further work [Jür00b] we exhibit conditions for the compositionality of secrecy using ideas from [Jür00c].

Future work will give internal characterisations for the notion of secrecy (that do not directly refer to adversaries and therefore are easier to check) and address other security properties such as integrity and authenticity and the integration into current work towards using the Unified Modeling Language to develop secure systems [Jür01].

8 Acknowledgements

Many thanks go to Martín Abadi for interesting discussions and to Zhenyu Qian for providing the opportunity to give a talk on this work at Kestrel Institute (Palo Alto) and to the members of the institute for useful feedback. Many thanks also to G. Wimmel and the anonymous referees for comments on the draft.

This work was performed during a visit at Bell Labs Research at Silicon Valley / Lucent Technologies (Palo Alto) whose hospitality is gratefully acknowledged.

References

[Aba00] M. Abadi. Security protocols and their properties. In F.L. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation*, pages 39–60. IOS Press, 2000. 20th Int. Summer School, Marktoberdorf, Germany.

[AG99] M. Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999.

[AJ00] M. Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation, 2000. submitted.

[APS99] V. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How much does it really cost ? In *Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.

[AR00] M. Abadi and P. Rogaway. Reconciling two views of cryptography (invited lecture). In *TCS 2000 (IFIP conference)*, Japan, August 2000.

[Boe81] B.W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[Bro99] M. Broy. A logical basis for component-based systems engineering. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. IOS Press, 1999.

[BS00] M. Broy and K. Stølen. *Specification and Development of Interactive Systems*. Springer, 2000. (to be published).

[CGG00] L. Cardelli, G. Ghelli, and A. Gordon. Secrecy and group creation. In *CONCUR 2000*, pages 365–379, 2000.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[FBGL94] S. Fitzgerald, T. M. Brookes, M. A. Green, and P. G. Larsen. Formal and informal specifications of a secure system component: first results in a comparative study. In M. Naftalin, B. T. Denvir, and M. Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, pages 35–44. Springer, 1994.

[GCS91] J. Graham-Cumming and J. Sanders. On the refinement of noninterference. In *IEEE Computer Security Foundations Workshop*, 1991.

[HMR+98] F. Huber, S. Molterer, A. Rausch, B. Schätz, M. Sihling, and O. Slotosch. Tool supported Specification and Simulation of Distributed Systems. In *International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 155–164, 1998.

[Jür00a] Jan Jürjens. Abstracting from failure probabilities, 2000. submitted.

[Jür00b] Jan Jürjens. Composability of secrecy, 2000. submitted.

[Jür00c] Jan Jürjens. Secure information flow for concurrent processes. In C. Palamidessi, editor, *CONCUR 2000 (11th International Conference on Concurrency Theory)*, volume 1877 of *LNCS*, pages 395–409, Pennsylvania, 2000. Springer.

[Jür01] Jan Jürjens. Towards development of secure systems using UML. In H. Hußmann, editor, *Fundamental Approaches to Software Engineering*, LNCS. Springer, 2001. to be published.

[LFBG95] P. G. Larsen, S. Fitzgerald, T. M. Brookes, and M. A. Green. Formal modelling and simulation in the development of a security-critical message processing system. In *Formal Methods, Modelling and Simulation for Systems Engineering*, 1995.

[Lot00] V. Lotz. Formally defining security properties with relations on streams. *Electronical Notes in Theoretical Computer Science*, 32, 2000.

[Man00] H. Mantel. Possibilistic definitions of security - an assembly kit. In *IEEE Computer Security Foundations Workshop*, 2000.

[McL94] J. McLean. Security models. In John Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons, Inc., 1994.

[McL96] J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.

[Mea92] C. Meadows. Using traces based on procedure calls to reason about composability. In *IEEE Symposium on Security and Privacy*, pages 177 – 188, 1992.

[Mea96] C. Meadows. Formal verification of cryptographic protocols: A survey. In *Asiacrypt 96*, 1996.

[Pfi98] B. Pfitzmann. Higher cryptographic protocols, 1998. Lecture Notes, Universität des Saarlandes.

[RS98] P. Ryan and S. Schneider. An attack on a recursive authentication protocol. *Inform. Proc. Letters*, 65:7–10, 1998.

[RS99] P. Ryan and S. Schneider. Process algebra and non-interference. In *IEEE Computer Security Foundations Workshop*, 1999.

[RWW94] A. Roscoe, J. Woodcock, and L. Wulf. Non-interference through determinism. In *ESORICS 94*, volume 875 of *LNCS*. Springer, 1994.

[Sch96] S. Schneider. Security properties and CSP. In *IEEE Symposium on Security and Privacy*, pages 174–187, 1996.

[SS75] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[SV00] P. Sewell and J. Vitek. Secure composition of untrusted code: Wrappers and causality types. In *CSFW*, 2000.