Composability of Secrecy

Jan Jürjens *

Computing Laboratory, University of Oxford, GB

Abstract. *Modularity* has been seen to be very useful in system development. Unfortunately, many security properties proposed in the literature are not composable (in contrast to other system properties), which is required to reason about them in a modular way.

We present work supporting modular development of secure systems by showing a standard notion of secrecy to be composable wrt. the standard composition in the specification framework Focus (extended with cryptographic primitives). Additionally, the property is preserved under the standard refinement. We consider more fine-grained conditions useful in modular verification of secrecy.

Keywords. Network security, cryptographic protocols, secrecy, modularity, composability, refinement, formal specification, computer aided software engineering.

This is the web-version 18/5/01 of a paper at Mathematical Methods, Models and Architectures for Computer Networks Security (MMM 2001), LNCS 2052, Springer Verlag. Please refer to http://www.jurjens.de/jan for the current version and other material.

1 Introduction

Mathematical models have been used extensively to ensure security of computer networks through reasoning about formal specifications [Mea96]. In spite of several success stories, major challenges remain. An important one of them is the problem of scaling up the used methods to large systems [Mea00].

For this, *modularity* is an important tool: One starts with considering components of a system that are so small that formal reasoning is still feasible. One proves that these components satisfy the required security property. If the security property under consideration is *composable* then (by definition) the whole system satisfies the security property.

Unfortunately, this is often not the case: various formulations of security properties in formal models are noncomposable (e.g. McLean's Applied Flow Model and Gray's Probabilistic Noninterference (in its original model) have been shown to be noncomposable in [Jür00]; exceptions are [Var91, Mea92]).

* jan@comlab.ox.ac.uk - http://www.jurjens.de/jan – This work was supported by the Studienstiftung des deutschen Volkes and the Computing Laboratory.

Not only does this make verification of specifications harder; even worse, it is also not clear what a non-composable security property guarantees in practice, where one would in fact like to connect systems to each other [Mil90].

In this work we consider composability of security properties, more specifically of *secrecy*. Continuing earlier work [Jür01b] we use an extension of the general-purpose, tool-supported CASE-framework Focus [BS00] with cryptographic operations including symmetric and asymmetric encryption and signing to consider composability of a notion of secrecy following a standard approach. We exhibit conditions under which it is composable and which make modular reasoning widely and usefully applicable. The secrecy property is also preserved under *refinement*, another highly useful strategy in formal system development which is often not given for security properties (*refinement paradox* [McL96]). This seems to be the first secrecy property shown to be preserved under composition and refinement. We also give more fine-grained notions of secrecy useful for modular development.

In the next subsection we give background on security and composability and refer to related work. In Section 2, we introduce the cryptographic extension of the specification framework Focus. In Section 3 we define the secrecy properties considered here. In Section 4 we consider composability and refinement. After that, we conclude.

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

1.1 Security and Composability

The approach of modular development (or "divide-and-conquer") has a long tradition in the formal development of systems. Therefore there has been some effort to show that properties of interest are composable. For example, Abadi and Lamport give a proof method for properties falling within the Alpern-Schneider framework of safety and liveness properties. Unfortunately, many security properties do not fall within the Alpern-Schneider framework [McL96].

Many secrecy properties follow one of the following two approaches (discussed in [Aba00]). 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, composability of it (also called the *hook-up property*) is related to the question if equivalence is a *congruence* which again is often not the case [RS99].

The secrecy property considered in this paper follows the second approach: 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 in clear, 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]. Since modularity is a highly useful property, a notion of secrecy that reveals only *most* weaknesses but is modular is well worth considering, especially since more fine-grained security properties may be hard to ensure in practice, as pointed out in [Mea95].

Related Work This line of work was initiated in [Jür01b], where secrecy was shown to be preserved under various standard refinements in the framework Focus and where it was used 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.

An overview on the use of formal methods in security protocols is given in [Mea96]. The need for composability is pointed out in [Mea00].

[Var91] gives a hook-up property for information flow secure nets. [Mea92] discusses composability of notions of secure information flow using traces based on procedure calls. [McL96] gives general results for composability of possibilistic notions of secure information flow. [Jür00] shows two notions of secure information flow, namely McLean's Applied Flow Model (AFM) and Gray's Probabilistic Noninterference (PNI) (in its original model) to be non-composable, and provides a slight modification of Gray's model where PNI is composable.

In [Lot97], threat scenarios are used to formally develop secure systems using Focus. Composability (and refinement) is left for further work. Further applications of Focus to system security are in [JW01].

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

2 Specification language

We give a short overview on the specification language used here; for a more detailed account cf. [Jür01b]. In this work, we view specifications as nondeterministic programs in the specification framework Focus [BS00] (providing mechanical assistance in form of the CASE tool Autofocus [HMR+98]). Executable specifications allow a rather straightforward modelling of cryptographic operators 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.

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 [Jür01b, AJ00]. 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 the error value $\perp$, 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 **Var** $\cup$ **Channels**). We write the decryption key corresponding to an encryption key K as K^{-1} (and call it a *private* key). 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 $E \in$ **Exp**, $K, K^{-1} \in$ **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 (where $E, E' \in$ **Exp** are expressions):

$p ::=$	programs
E	output expression
either p *or* p'	nondeterministic branching
if $E = E'$ *then* p *else* p'	conditional
case E *of key do* p *else* p'	determine if E is a key
case E *of* $x :: y$ *do* p *else* p'	break up list into head and tail

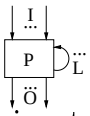
Variables are introduced in case constructs, which determine their values. The first case construct tests whether E is a key; if so, p is executed, otherwise p'. The second case construct tests whether E is a list with head x and tail y; if so, p is evaluated, using the actual values of x, y; if not, 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 $c \in$ **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 each decryption $\mathcal{D}ec_e(E)$ appears as part of p in a *case* E' *of key do* p *else* p' construct (unless $e \in$ **Keys**), to ensure that only keys are used to encrypt or decrypt. It is straightforward to enforce this using a type system.

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, $K_P \subseteq \mathbf{Keys}$ for the set of the *private* keys and $S_P \subseteq \mathbf{Secret}$ for the set of unguessable values (such as nonces) occurring in P. We assume that different processes have disjoint sets of local channels.

2.1 Stream-processing functions

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

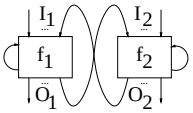
We write $\mathbf{Stream}_C \stackrel{\text{def}}{=} (\mathbf{CExp}^\omega)^C$ (where $C \subseteq \mathbf{Channels}$) for the set of C-indexed tuples of (finite or infinite) sequences of closed expressions. The elements of this set are called *streams*, specifically *input streams* (resp. *output streams*) 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 $c \in C'$. 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.

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

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

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

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).

We define composition of the (non-rushing) processes $P = (I_P, O_P, L_P, (p_c)_{c \in O_P \cup L_P})$ and $Q = (I_Q, O_Q, L_Q, (p_c)_{c \in O_Q \cup L_Q})$ with $O_P \cap O_Q = L_P \cap L_Q = \emptyset$ to be

$$P \otimes Q \stackrel{\text{def}}{=} (I_{P \otimes Q}, O_{P \otimes Q}, L_{P \otimes Q}, (p_c)_{c \in O_{P \otimes Q} \cup L_{P \otimes Q}})$$

where $I_{P \otimes Q} = (I_P \cup I_Q) \setminus (O_P \cup O_Q)$, $O_{P \otimes Q} = (O_P \cup O_Q) \setminus (I_P \cup I_Q)$ and $L_{P \otimes Q} = L_P \cup L_Q \cup ((I_P \cup I_Q) \cap (O_P \cup O_Q))$.

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 $(\boldsymbol{t}(c))_j = E$. When we say that f may eventually output an expression we implicitely assume that f has no input channels; in particular, if e.g. $f = [\![P]\!] \otimes [\![Q]\!]$, we assume that $I_P \subseteq O_Q$ and $I_Q \subseteq O_P$ (and $O_P \cap O_Q = L_P \cap L_Q = \emptyset$ for composition to be well-defined).

Definition 1. *We say that a process P* preserves *a secret $m \in \mathbf{Secret} \cup \mathbf{Keys}$ if there is no process A with $m \notin S_A \cup K_A$ and $K_P \cap K_A = \emptyset$ such that $[\![P]\!] \otimes [\![A]\!]_r$ may eventually output m.*

The idea of this definition is that P preserves the secrecy of m if no adversary who does not already know the private keys of P 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 $P \stackrel{\text{def}}{=} (\{c\}, \{e\}, (p_d, p_e))$ (where $p_e \stackrel{\text{def}}{=} \{l\}_K$) does.

One can also define a rely-guarantee version of secrecy as in [Jür01b].

4 Composability

Definition 2. *A process* P respects secrecy of $m \in \mathbf{Keys} \cup \mathbf{Secret}$ *if for all processes* Q *that preserve the secrecy of* m *and all processes* A *with* $m \notin S_A \cup K_A$ *and* $K_Q \cap K_A = \emptyset$ *such that* $[\![P]\!] \otimes [\![Q]\!] \otimes [\![A]\!]_r$ *may eventually output* m*, there is a process* A' *with* $m \notin S_{A'} \cup K_{A'}$ *and* $K_Q \cap K_{A'} = \emptyset$ *such that* $[\![A']\!]_r \otimes [\![Q]\!] \otimes [\![A]\!]_r$ *may eventually output* m*.*

Example The process P that tries to decrypt any input with K and outputs the result if successful, does not respect the secrecy of any m.

Theorem 1. *Suppose that the processes* P *and* P' *preserve and respect secrecy of* m*. Then* $P \otimes P'$ *preserves and respects secrecy of* m*.*

It is necessary to assume that P and P' respect secrecy of m: Consider a process P that outputs m encrypted under K and a process P' that tries to decrypt any input with K and outputs the result if successful. Both P and P' preserve secrecy of m, but $P \otimes P'$ does not.

For compositionality considerations it is very useful to have a more fine-grained notion of preservation of secrecy.

Definition 3. *Suppose we are given a process* P*, a value* $m \in \mathbf{Keys} \cup \mathbf{Secret}$ *and a set of private keys* $\mathcal{K} \subseteq \mathbf{Keys}$*. We say that*

- P protects the secrecy of m with the keys in $\mathcal{K}$ *if for any process* A *with* $m \notin S_A \cup K_A$ *such that* $[\![P]\!] \otimes [\![A]\!]_r$ *may eventually output* m*, we have* $\mathcal{K} \subseteq K_A$*, and*
- P respects the protection of m by $\mathcal{K}$ *if for all processes* Q *that protect the secrecy of* m *with* $\mathcal{K}$ *and all processes* A *with* $m \notin S_A \cup K_A$ *and* $\mathcal{K} \not\subseteq K_A$ *such that* $[\![P]\!] \otimes [\![Q]\!] \otimes [\![A]\!]_r$ *may eventually output* m*, there is a process* A' *with* $m \notin S_{A'} \cup K_{A'}$ *and* $\mathcal{K} \cap K_{A'} = \emptyset$ *such that* $[\![A']\!]_r \otimes [\![Q]\!] \otimes [\![A]\!]_r$ *may eventually output* m*.*

The idea is that an adversary can find out the secret only if she has access to the keys in $\mathcal{K}$. One can give an even more fine-grained definition saying that P *protects the secrecy of* m *with the combinations of keys in* $\mathcal{K}$ for a set $\mathcal{K} \subseteq \mathcal{P}(\mathbf{Keys})$ of key sets, where the idea is that an adversary can find out the secret only if she has access to one of the sets of the keys in $\mathcal{K}$. Then P preserves the secrecy of m if and only if P protects the secrecy of m with the combination of keys in $\mathcal{K}$ consisting of all singleton sets $\{k\}$ with $k \in K_P$.

Examples

- $p \stackrel{\text{def}}{=} \{m\}_K :: K$ does not protect the secrecy of m with $\{K\}$, but $p \stackrel{\text{def}}{=} \{m\}_K$ does.
- The process that tries to decrypt any input with K and outputs the result if successful, does not respect the protection of any m by $\{K\}$, but the process that tries to decrypt any input with K and outputs the result if successful respects the protection of m_0 by $\{K'\}$ (for $K' \neq K$).

Theorem 2. *Suppose that the processes P and P' protect the secrecy of m with $\mathcal{K}$ and that they respect the protection of m by $\mathcal{K}$. Then $P \otimes P'$ protects the secrecy of m with $\mathcal{K}$ and respects the protection of m by $\mathcal{K}$.*

To apply this result, we consider an intensional version of secrecy protection.

Definition 4. *A process P* protects m intensionally with the combinations of keys in $\mathcal{K}$ *if there exists a set of local channels $H_P \subseteq L_P$ such that*

- *an output p_c to a channel c may contain subexpressions of the form $\mathcal{D}ec_e(E)$ only if*
  - *$c \in H_P$, or*
  - *e is protected (by suitable conditionals) from being evaluated at run-time to a key in the union $\bigcup_{\mathbf{K} \in \mathcal{K}} \mathbf{K}$ of key sets in $\mathcal{K}$, or*
  - *E is protected from containing m as a subexpression,*
- *an output p_c to a channel may contain a subexpression m or d (for a channel $d \in H_P$) only if*
  - *$c \in H_P$, or*
  - *the subexpression occurs as the key used in an encryption or decryption, or*
  - *it is encrypted under all keys in one of the sets $\mathbf{K} \in \mathcal{K}$.*

One can enforce these conditions with a type system; this has to be left out here.

One may also partition the set of **Channels** into a set of trusted and a set of untrusted channels (e.g. the communication channels within a system and those across a network). Then the above definition can be extended by requiring the set H_P to include the trusted channels.

Theorem 3. *If a process P protects m intensionally with the combinations of keys in $\mathcal{K}$ then P protects m with the combinations of keys in $\mathcal{K}$.*

Refinement We define the standard refinement from [BS00] and show that it preserves secrecy. Further refinement operators are considered in [Jür01b].

Definition 5. *For processes P and P' with $I_P = I_{P'}$ and $O_P = O_{P'}$ we define $P \rightsquigarrow P'$ if for each $\boldsymbol{s} \in \mathbf{Stream}_{I_P}$, $[\![P]\!](\boldsymbol{s}) \supseteq [\![P']\!](\boldsymbol{s})$.*

Theorem 4. *For each of the secrecy properties p defined above (possibly wrt. m or $\mathcal{K}$) the following implication holds:*

If P satisfies p and $P \rightsquigarrow P'$ then P' satisfies p.

5 Conclusion and Future Work

We presented work towards a framework for modular development of secure systems by showing a notion of secrecy (that follows a standard approach) to be composable and preserved under refinement in the specification framework Focus extended by cryptographic primitives. We gave more fine-grained notions of secrecy that are useful for verification purposes.

Future work will 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 [Jür01c, Jür01a].

Acknowledgements Many thanks go to Martín Abadi for interesting discussions. Many thanks also to Zhenyu Qian for providing the opportunity to give a talk on an early version of this work at Kestrel Institute (Palo Alto) and to the members of the institute for useful feedback. Part of 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. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation*. IOS Press, 2000.

[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.

[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.

[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.

[Jür00] Jan Jürjens. Secure information flow for concurrent processes. In *CONCUR 2000 (11th International Conference on Concurrency Theory)*, volume 1877 of *LNCS*, pages 395–409, Pennsylvania, 2000. Springer.

[Jür01a] Jan Jürjens. Object-oriented modelling of audit security for smart-card payment schemes. In P. Paradinas, editor, *IFIP/SEC 2001 – 16th International Conference on Information Security*, Paris, 11–13 June 2001. Kluwer.

[Jür01b] Jan Jürjens. Secrecy-preserving refinement. In *Formal Methods Europe (International Symposium)*, LNCS. Springer, 2001.

[Jür01c] Jan Jürjens. Towards development of secure systems using UML. In H. Hußmann, editor, *Fundamental Approaches to Software Engineering (FASE/ETAPS, International Conference)*, LNCS. Springer, 2001.

[JW01] Jan Jürjens and Guido Wimmel. Specification-based testing of firewalls. Submitted, 2001.

[Lot97] V. Lotz. Threat scenarios as a means to formally develop secure systems. *Journal of Computer Security 5*, pages 31–67, 1997.

[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*, 1992.

[Mea95] C. Meadows. Applying the dependability paradigm to computer security. In *New Security Paradigms Workshop*, 1995.

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

[Mea00] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *DISCEX*. IEEE, 2000.

[Mil90] J. Millen. Hookup security for synchronous machines. In *Computer Security Foundations Workshop III*. IEEE Computer Society, 1990.

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

[PW93] H. Pohl and G. Weck, editors. *Internationale Sicherheitskriterien*. Oldenbourg Verlag, 1993.

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

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

[Var91] V. Varadharajan. Hook-up property for information flow secure nets. In *CSFW*, 1991.