T U M

I N S T I T U T F Ü R I N F O R M A T I K

Stream Based Specification of Cryptographic Protocols and Their Composition Properties

Maria Spichkova, Jan Jürjens

TUM-I0823
Juli 08

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-07-I0823-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Stream Based Specification of Cryptographic Protocols and Their Composition Properties

Maria Spichkova[1]   Jan Jürjens[2]

[1] Institut für Informatik,Technische Universität München
Boltzmannstr. 3, D-85748 Garching, Germany
http://www4.in.tum.de/~spichkov

[2] Computing Department, The Open University
Milton Keynes, MK7 6AA, Great-Britain
http://mcs.open.ac.uk/jj2924

Abstract

The correct development of security-critical systems is very difficult, as demonstrated by many insecure systems that have been developed in research and practice. A particular challenge is the establishment of security properties for separate components in an open, distributed system, in a way that the interaction of these components will still satisfy the security properties established for each component in isolation.

We present a methodology to represent crypto-based, distributed software (such as cryptographic protocols) and their composition properties in a formal way using Focus, a framework for formal specification and development of interactive systems. Using this formal representation, one can argue about properties of protocol components and their composition in a methodological way. We use the Focus approach, because it was developed specifically to support the compositional development of distributed systems and offers a number of specification techniques including several practical notions of refinement. It also supports formal arguments about property combination using well-founded theories of component- and service-composition.

Keywords: Formal Specification, Verification, Cryptographic Protocols, Protocols Properties

---

 This work was partially supported by the Royal Society within the project Model-based Formal Security Analysis of Crypto-Protocol Implementations.

Contents

1 Introduction

Developing security-critical systems is one of the most challenging fields of systems engineering. Especially difficult is the question how we can combine system components that each enforce a particular security requirement in a way that allows us to predict which properties the combined system will have. For this purpose we have developed a methodology to represent crypto-based software (such as cryptographic protocols) and their composition properties in a formal way.

Having a verified formal specification we can be sure that the specification conforms to its requirements and is consistent. We use the approach Focus [7], a framework for formal specification and development of interactive systems. As a running example, we use a variant of the Internet security protocol TLS published in [3]. Using our approach, we demonstrate a security flaw in the protocol, and show how to prove security properties of a corrected version, as well as how to formally establish a secure channel using the secured version. We also provide some general results on composition of security properties.

Besides being a useful specification and development approach in its own right, the formal approach presented in this paper also serves as a formal foundation for other approaches using widely used development approaches, specifically the approach for developing secure software using the UML extension UMLsec presented in [14]. Using the formal semantics for the fragment of UML used in UMLsec which is presented on the basis of Focus in [14], we can use the approach presented in this paper to reason formally about UMLsec specification and their composition.

Our approach also prepares the ground for the possibility to verify the specifications against security properties by translating them to the theorem prover Isabelle/HOL [18] using the framework "Focus on Isabelle" [19] (this will be explained in detail in a subsequent paper).

Using our approach, we can influence the complexity of proofs and their reusability already during the specification phase, because the specification and verification/validation methodologies are treated here as a single joint methodology with the main focus on the specification part. Moreover, using the framework "Focus on Isabelle" one can perform automatic correctness proofs of syntactic interfaces for specified system components.

2 Focus

Focus [7] is a framework for formal specifications and development of distributed interactive systems. A system in Focus is represented by its components that are connected by communication lines called *channels*, and are described in terms of its input/output behavior. The components can interact and also work independently of each other. A specification can be elementary or composite – composite specifications are built hierarchically from the elementary ones.

The channels in this specification framework are *asynchronous communication links* without delays. They are *directed* and generally assumed to be *reliable*, and *order preserving* (although in the next section we explain how to modify this to allow of interference by attackers on the network). Via these channels components exchange information in terms of *messages* of specified types. Messages are passed along the channels one after the other and delivered in exactly the same order in which they were sent (unless there is some attacker interaction; cf. next section).

In Focus any specification characterizes the relation between the *communication histories* for the external *input* and *output channels*. To denote that the (lists of) input and output channel identifiers, I and O, build the syntactic interface of the specification S the notation $(I_P \triangleright O_P)$ is used. The formal meaning of a specification is exactly this external *input/output relation*.

The Focus specifications can be structured into a number of formulas each characterizing a different kind of property, the most prominent classes of them are *safety* and *liveness properties*. Focus supports a variety of *specification styles* which describe system components by logical formulas or by diagrams and tables representing logical formulas.

The central concept in Focus are *streams*, that represent communication histories of *directed channels*. For any set of messages M, M^ω denotes the set of all streams, M^∞ and M^* denote the sets of all infinite and all finite streams respectively, $M^{\underline{\omega}}$ denotes the set of all timed streams, $M^{\underline{\infty}}$ and $M^{\underline{*}}$ denote the sets of all infinite and all finite timed streams respectively. A *timed stream* is represented by a sequence of messages and *time ticks*, the messages are also listed in their order of transmission. The ticks model a discrete notion of time. The notion of time provided by the timed streams allows us to correctly specify system components, and to compose them with the anomalies that may occur in the untimed treatment (Brock-Ackermann anomaly).

The specification scheme of Focus supports a variety of specification styles which describe system components by logical formulas or by diagrams and tables representing logical formulas. It has an integrated notion of time and modeling techniques for unbounded networks, provides a number of specification techniques for distributed systems and concepts of refinement.

The Focus specification framework uses three basic refinement relations: behavioral, interface and conditional refinement. We are using here the definitions of the behavioral refinement from [7]: A specification S_2 is called a *behavioral refinement* ($S_1 \rightsquigarrow S_2$) of a specification S_1 if they have the same syntactic interface and any I/O history of S_2 is also an I/O history of S_1. Formally, we need to show that any I/O history of S_2 is an I/O history of S_1, but S_1 may have additional I/O histories. When verifying Focus specifications using Isabelle/HOL, this means that one needs to prove that the formula that corresponds to the semantics of the specification body $[\![S_2]\!]$ implies the formula that corresponds to $[\![S_1]\!]$.

The most general style of a Focus specification is an A/G style (Assumption/Guarantee style, Assumption/Committment style) – a component is specified in terms of an assumption and a guarantee, what means whenever input from the environment behaves in accordance with the assumption asm, the specified component is required to fulfill the guarantee gar. We suggest to use this style in the most cases. The only exception is the pure system architecture specification, which serves only to show in a readable way how the subcomponents are connected. If for some component we have not any assumption, we can also fill the assumption part with true. In such a way we can partially solve the problem with forgotten assumptions.

Focus operators used in the paper:
An empty stream is represented in FOCUS by $\langle\rangle$.
$\langle x\rangle$ denotes the one element stream consisting of the element x.
$\#s$ denotes the length of the stream s.
ith time interval of the stream s is represented by $\mathsf{ti}(s,i)$.
$\mathsf{msg}_n(s)$ denotes a stream s that can have at most n messages at each time interval.
s^i_{ft}, s^i_{snd} and s^i_{trd} denote the first, the second and the third elements of the ith time interval of the stream s respectively (partial functions).

See [7] and [19] for more background on FOCUS and its extensions.

3 Composing Protocol Components

By representing protocols as FOCUS specifications, we can describe them as components or services (see [7, 9]) and can argue about properties of component compositions using well-founded theories of component- and service-composition (see [6, 8]).

The FOCUS semantics of a *composite* specification $S = S_1 \otimes \cdots \otimes S_n$ is defined in [7] as follows:

$$[\![\ S\ ]\!] \stackrel{\mathsf{def}}{=} \exists\, l_S \in L_S : \bigwedge_{j=1}^{n} [\![\ S_j\ ]\!] \tag{1}$$

where l_S denotes a set of *local streams* and L_S denotes their corresponding types, $[S_j]$ denotes semantics of the FOCUS specification S_j, $1 \leq j \leq n$, which is a specification of subcomponent of S.
For any FOCUS specification S the sets i_S and o_S must be disjoint:

$$i_S \cap o_S = \varnothing \tag{2}$$

For any composite FOCUS specification S the sets i_S, o_S and l_S must be pairwise disjoint, i.e. the following equations must hold:

$$\begin{aligned} i_S \cap l_S &= \varnothing \\ l_S \cap o_S &= \varnothing \end{aligned} \tag{3}$$

Equation 3 trivially holds for any elementary specification, because for any elementary specification S the set l_S is empty. Thus, Equations 2 and 3 build together the common property of correct relations between the sets of input, output and local channels.

The sets i_S and o_S of input and output channel identifiers of a composite specification S consist of all sets of input and output channel identifiers of composing specifications $S_1, \ldots, S_n$ excluding the channels which are used for the local communication:

$$i_S \stackrel{\mathsf{def}}{=} \bigcup_{j=1}^{n} (i_{S_j} \in I_S^{\infty}) \setminus l_S \tag{4}$$

$$o_S \stackrel{\mathsf{def}}{=} \bigcup_{j=1}^{n} (o_{S_j} \in O^{\infty}_{S_j}) \setminus l_S \tag{5}$$

These equations imply also the following ones:

$$i_S \subseteq \bigcup_{j=1}^{n} (i_{S_j} \in I^{\infty}_{S}) \tag{6}$$

$$o_S \subseteq \bigcup_{j=1}^{n} (o_{S_j} \in O^{\infty}_{S_j}) \tag{7}$$

For the specification of the system S that is composed from the specifications $S_1, \ldots, S_n$ the following properties must hold [19]:

- For the set of input streams of the system S:
  Equation 4 holds. No input stream i can be an output stream of any subcomponent.

$$i_S = \bigcup_{j=1}^{n} (i_{S_j} \in I^{\infty}_{S}) \setminus l_S \ \wedge \ i_S \cap \bigcup_{j=1}^{n} o_{S_j} = \varnothing \tag{8}$$

- For the set of output streams of the system S:
  Equation 5 holds. No output stream i of the system S can be an input stream of any subcomponent.

$$o_S = \bigcup_{j=1}^{n} (o_{S_j} \in O^{\infty}_{S_j}) \setminus l_S \ \wedge \ o_S \cap \bigcup_{j=1}^{n} i_{S_j} = \varnothing \tag{9}$$

- Every local stream l of the system S must be both an input stream of some subcomponent S_{j_1}, $1 \leq j_1 \leq n$, and an output stream of some subcomponent S_{j_2}, $1 \leq j_2 \leq n$ $(j_1 \neq j_2)$:

$$l_S = \bigcup_{j=1}^{n} i_{S_j} \cap \bigcup_{j=1}^{n} o_{S_j} \tag{10}$$

We can thus combine different components involved in a protocol (see Section 5) and can check whether this combination satisfies the desired security properties. There are different kinds of compositions in Focus, such as composition by feedback $\mu\, F$, parallel composition $(F_1 \,||\, F_2)$, and sequential composition $(F_1;F_2)$.

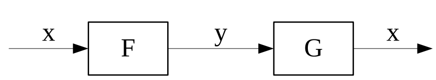


Figure 1: Sequential Composition

We denote by $\mathsf{subcomp}(P)$ the set of subcomponents of a component P: for a composite component S

$$S = S_1 \otimes \cdots \otimes S_n$$

we get

$$\mathsf{subcomp}(S) = \{S_1, \ldots, S_n\}$$

If P is an elementary component, the set $\mathsf{subcomp}(P)$ will be empty.

We discuss one such case of composition. Two components, F and G (e.g. the client parts of two different protocols where one of them is layered on top of the other) need to be combined sequentially, which in FOCUS is specified as follows: $(F;G)$. Let us assume that the component F has one input channel $x \in M_1$ and one output channel $y_1 \in M_2$, and that the component G has one input channel $y_2 \in M_3$ and one output channel $x \in M_4$, where M_1, M_2, M_3 and M_4 are some data types of the corresponding streams. The component F specifies some stream processing function $f_F : M_1{}^{\underline{\omega}} \rightarrow M_2{}^{\underline{\omega}}$, s.t. $y_1 = f_F(x)$. The component G specifies another stream processing function $f_G : M_3{}^{\underline{\omega}} \rightarrow M_4{}^{\underline{\omega}}$, s.t. $z = f_F(y_2)$. Combining the components F and G sequentially, we get $z = f_G(f_F(x))$ and we can define the corresponding stream processing function $f_{FG} : M_1{}^{\underline{\omega}} \rightarrow M_4{}^{\underline{\omega}}$.

Note that for this composition to be well-defined, a number of formal constraints need to be satisfied, e.g. the data type M_2 must be equal to the data type M_3. Moreover, if the component G has some assumption about the data stream y_2, these properties must hold for the data stream y_1 – if they do not hold, the composition is not well-defined. E.g., if the assumption part of the specification of the component G contains $\mathsf{ts}(y_2)$, this property must hold for the stream y_1 of the component F – either this predicate must belong to the guarantee part of the specification of the component F, or it must be possible to prove from the guarantee part of the specification that $\mathsf{ts}(y_1)$ holds.

Thus, we can reduce the problem of protocol component composition to the problem of function (or component/service-) composition. This also means that when specifying a protocol component, one needs to analyze the preconditions of its correct activity and specify them in the assumption part. Missing assumptions and incompatibilities of properties will be detected during the verification. For this purpose we can translate the FOCUS specification into Isabelle/HOL and verify them using the methodology "FOCUS on Isabelle"[19].

A number of propositions and theorems about the security properties of composed systems are presented in Section 4.

4 Secrecy

In this section we introduce a formalization of the security property of data secrecy, the corresponding definitions, and a number of abstract data types used in this formalization.

4.1 Data Types

We assume disjoint sets *Data* of data values, *Secret* of unguessable values, and *Keys* of cryptographic keys. Based on these sets, we specify the sets *EncType* of *encryptors* that may be used for encryption or decryption, *CExp* of closed expressions, and *Expression* of expression items:

$$
\begin{aligned}
&\mathit{KS} && \stackrel{\mathsf{def}}{=} && \mathit{Keys} \cup \mathit{Secret}\\
&\mathit{EncType} && \stackrel{\mathsf{def}}{=} && \mathit{Keys} \cup \mathit{Var}\\
&\mathit{CExp} && \stackrel{\mathsf{def}}{=} && \mathit{Data} \cup \mathit{Keys} \cup \mathit{Secret}\\
&\mathit{Expression} && \stackrel{\mathsf{def}}{=} && \mathit{Data} \cup \mathit{Keys} \cup \mathit{Secret} \cup \mathit{Var}
\end{aligned}
$$

Below, we will treat an *expression* (that can for example be sent as an argument of a message within the distributed system) as a finite sequence of expression items. $\langle\rangle$ then denotes an empty expression.

The decryption key corresponding to an encryption key K is written as K^{-1}. In the case of asymmetric encryption, the encryption key K is public, and the decryption key K^{-1} secret. For symmetric encryption, K and K^{-1} coincide. For the encryption, decryption, signature creation and signature verification functions we define only their signatures and general axioms, because in order to reason effectively, we view them as abstract functions and abstract from their bit-level implementation details (following the usual Dolev-Yao approach to crypto-protocol verification [12]):

$$
\begin{aligned}
&\mathit{Enc} :: \mathit{EncType} \times \mathit{Expression}^* \to \mathit{Expression}^*\\
&\mathit{Decr} :: \mathit{EncType} \times \mathit{Expression}^* \to \mathit{Expression}^*\\
&\mathit{Sign} :: \mathit{EncType} \times \mathit{Expression}^* \to \mathit{Expression}^*\\
&\mathit{Ext} :: \mathit{EncType} \times \mathit{Expression}^* \to \mathit{Expression}^*\\
&\forall\, e \in \mathit{Expression} :\\
&\qquad \mathit{Ext}(K, \mathit{Sign}(K^{-1}, e)) = e\\
&\qquad \mathit{Decr}(\mathit{CKey}^{-1}, \mathit{Enc}(\mathit{CKey}, e)) = e
\end{aligned}
$$

We denote by $K_P \subseteq \mathit{Keys}$ and $S_P \subseteq \mathit{Secret}$ the set of private keys of a component P and the set of unguessable values used by a component P, respectively. The union of these two sets will be denoted by KS_P.

The sets of private keys and unguessable values used by a composed component $C = C_1 \otimes \cdots \otimes C_n$ is defined by union of corresponding sets:

$$
\begin{aligned}
&K_C = K_{C_1} \cup \ldots \cup K_{C_n}\\
&S_C = S_{C_1} \cup \ldots \cup S_{C_n}\\
&KS_C = KS_{C_1} \cup \ldots \cup KS_{C_n}
\end{aligned}
$$

4.2 Input and Output of Expressions

We say that a component P, $(I_P \triangleright O_P)$, may eventually output an expression $E \in \mathit{CExp}$ (denoted by $P^{\mathsf{eout}}(E)$), if there exists a time interval t of an output stream $s \in o_P$ which contains this expression E:

$$P^{\mathsf{eout}}(E) \stackrel{\mathsf{def}}{=} \exists\, s \in o_P : \ \exists\, t \in \mathbb{N} : \ E \in \mathsf{ti}(s, t)$$

A component P, $(I_P \rhd O_P)$, may eventually output an expression $E \in CExp$ via M (denoted by $P_M^{\mathsf{eout}}(E)$) if M is the set of channels, which is a subset of output channels of the component P ($M \subseteq o_P$), and if there exists a time interval t of a stream $s \in M$ which contains this expression E:

$$P_M^{\mathsf{eout}}(E) \stackrel{\mathsf{def}}{=} M \subseteq o_P \ \wedge \ \exists\, s \in M : \ \exists\, t \in \mathbb{N} : \ E \in \mathsf{ti}(s,t)$$

A component P, $(I_P \rhd O_P)$, may eventually get an expression $E \in CExp$ (denoted by $P^{\mathsf{ine}}(E)$), if there exists a time interval t of an input stream $s \in i_P$ which contains this expression E:

$$P^{\mathsf{ine}}(E) \stackrel{\mathsf{def}}{=} \exists\, s \in i_P : \ \exists\, t \in \mathbb{N} : \ E \in \mathsf{ti}(s,t)$$

A component P, $(I_P \rhd O_P)$, may eventually get an expression $E \in CExp$ via M (denoted by $P_M^{\mathsf{ine}}(E)$) if M is the set of channels, which is a subset of input channels of the component P, and if there exists a time interval t of a stream $s \in M$ which contains this expression E:

$$P_M^{\mathsf{ine}}(E) \stackrel{\mathsf{def}}{=} M \subseteq i_P \ \wedge \ \exists\, s \in M : \ \exists\, t \in \mathbb{N} : \ E \in \mathsf{ti}(s,t)$$

Remark: Please note, that in the definitions of $P^{\mathsf{eout}}(E)$, $P_M^{\mathsf{eout}}(E)$, $P^{\mathsf{ine}}(E)$ and $P_M^{\mathsf{ine}}(E)$ we actually need to take into account only those streams, which are of type *Expression* or whose type contains the type *Expression*.

Theorem 1 *For any components P and Q the composition $P \otimes Q$ has the following properties ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$):*

$$(P \otimes Q)^{\mathit{ine}}(e) \ \rightarrow \ P^{\mathit{ine}}(e) \ \vee \ Q^{\mathit{ine}}(e) \qquad (1)$$
$$(P \otimes Q)_M^{\mathit{ine}}(e) \ \rightarrow \ P_M^{\mathit{ine}}(e) \ \vee \ Q_M^{\mathit{ine}}(e) \qquad (2)$$

Proof:
By the definition of ine *we have:*

$$P^{\mathit{ine}}(e) \ \vee \ Q^{\mathit{ine}}(e)$$
$$\equiv$$
$$\exists\, s_1 \in i_P : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s_1,t) \ \vee \ \exists\, s_2 \in i_Q : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s_2,t)$$
$$\equiv$$
$$\exists\, s \in (i_P \cup i_Q) : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s,t)$$

$$(P \otimes Q)^{\mathit{ine}}(e)$$
$$\equiv$$
$$\exists\, s \in i_{P \otimes Q} : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s,t)$$

By Equation 6 we have that $i_{P \otimes Q} \subseteq (i_P \cup i_Q)$, *i.e. that*

$$\exists\, s \in i_{P \otimes Q} : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s,t)$$
$$\Rightarrow$$
$$\exists\, s \in (i_P \cup i_Q) : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s,t)$$

The proof for $P_M^{\mathsf{ine}}(e)$ *is analogous.* □

Theorem 2 *For any components* P *and* Q *the composition* $P \otimes Q$ *has the following properties (*$e \in Expression$, $m \in KS$, $m \notin KS_P$ *and* $m \notin KS_Q$*):*

$$(P \otimes Q)^{\mathsf{eout}}(e) \ \rightarrow \ P^{\mathsf{eout}}(e) \ \vee \ Q^{\mathsf{eout}}(e) \qquad (1)$$
$$(P \otimes Q)_M^{\mathsf{eout}}(e) \ \rightarrow \ P_M^{\mathsf{eout}}(e) \ \vee \ Q_M^{\mathsf{eout}}(e) \qquad (2)$$

Proof:
By the definition of eout *we have:*

$$P^{\mathsf{eout}}(e) \ \vee \ Q^{\mathsf{eout}}(e)$$
$$\equiv$$
$$\exists\, s_1 \in o_P : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s_1,t) \ \vee \ \exists\, s_2 \in o_Q : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s_2,t)$$
$$\equiv$$
$$\exists\, s \in (o_P \cup o_Q) : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s,t)$$

$$(P \otimes Q)^{\mathsf{eout}}(e)$$
$$\equiv$$
$$\exists\, s \in o_{P \otimes Q} : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s,t)$$

By Equation 7 we have that $o_{P \otimes Q} \subseteq (o_P \cup o_Q)$, *i.e. that*

$$\exists\, s \in o_{P \otimes Q} : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s,t)$$
$$\Rightarrow$$
$$\exists\, s \in (o_P \cup o_Q) : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s,t)$$

The proof for $P_M^{\mathsf{eout}}(e)$ *is analogous.* □

Theorem 3 *For any components* P *and* Q *the composition* $P \otimes Q$ *has the following properties (*$e \in Expression$, $m \in KS$, $m \notin KS_P$ *and* $m \notin KS_Q$*):*

$$\neg P^{\mathsf{ine}}(e) \ \wedge \ \neg Q^{\mathsf{ine}}(e) \ \rightarrow \ \neg (P \otimes Q)^{\mathsf{ine}}(e) \qquad (1)$$
$$\neg P_M^{\mathsf{ine}}(e) \ \wedge \ \neg Q_M^{\mathsf{ine}}(e) \ \rightarrow \ \neg (P \otimes Q)_M^{\mathsf{ine}}(e) \qquad (2)$$

Proof:
By the definition of ine *we have:*

$$\neg P^{\mathsf{ine}}(e) \ \wedge \ \neg Q^{\mathsf{ine}}(e)$$
$$\equiv$$
$$\neg(\exists\, s_1 \in i_P : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s_1, t)) \ \wedge \ \neg(\exists\, s_2 \in i_Q : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s_2, t))$$
$$\equiv$$
$$\forall\, s_1 \in i_P : \ \forall\, t \in \mathbb{N} : \ e \notin \mathit{ti}(s_1, t) \ \wedge \ \forall\, s_2 \in i_Q : \ \forall\, t \in \mathbb{N} : \ e \notin \mathit{ti}(s_2, t)$$
$$\equiv$$
$$\forall\, s \in (i_P \cup i_Q) : \ \forall\, t \in \mathbb{N} : \ e \notin \mathit{ti}(s, t)$$

$$\neg(P \otimes Q)^{\mathsf{ine}}(e)$$
$$\equiv$$
$$\neg(\exists\, s \in i_{P\otimes Q} : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s, t))$$
$$\equiv$$
$$\forall\, s \in i_{P\otimes Q} : \ \forall\, t \in \mathbb{N} : \ e \notin \mathit{ti}(s, t)$$

By Equation 6 we have that $i_{P\otimes Q} \subseteq (i_P \cup i_Q)$*, i.e. that*

$$\forall\, s \in (i_P \cup i_Q) : \ \forall\, t \in \mathbb{N} : \ e \notin \mathit{ti}(s, t)$$
$$\Rightarrow$$
$$\forall\, s \in i_{P\otimes Q} : \ \forall\, t \in \mathbb{N} : \ e \notin \mathit{ti}(s, t)$$

The proof for $P_M^{\mathsf{ine}}(e)$ *is analogous.* □

Theorem 4 *For any components* P *and* Q *in general the following properties of the composition* $P \otimes Q$ *(*$e \in Expression$*,* $m \in KS$*,* $m \notin KS_P$ *and* $m \notin KS_Q$*) does NOT hold:*

$$P^{\mathsf{ine}}(e) \ \vee \ Q^{\mathsf{ine}}(e) \ \rightarrow \ (P \otimes Q)^{\mathsf{ine}}(e)$$
$$P_M^{\mathsf{ine}}(e) \ \vee \ Q_M^{\mathsf{ine}}(e) \ \rightarrow \ (P \otimes Q)_M^{\mathsf{ine}}(e)$$

Proof:
By the definition of ine *we have:*

$$P^{\mathsf{ine}}(e) \ \vee \ Q^{\mathsf{ine}}(e)$$
$$\equiv$$
$$(\exists\, s_1 \in i_P : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s_1, t)) \ \vee \ (\exists\, s_2 \in i_Q : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s_2, t))$$
$$\equiv$$
$$\exists\, s \in (i_P \cup i_Q) : \ \exists\, t \in \mathbb{N} : \ e \in \mathit{ti}(s, t)$$

$(P \otimes Q)^{\mathsf{ine}}(e)$

$\equiv$

$\exists\, s \in i_{P \otimes Q} : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s, t)$

By Equation 6 we have that $i_{P \otimes Q} \subseteq (i_P \cup i_Q)$, *i.e. in general we can have some stream* $s \in (i_P \cup i_Q)$ *for which* $\exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s, t)$ *holds, but this stream does not necessary belongs to the set* $i_{P \otimes Q}$.

The proof for $P_M^{\mathsf{ine}}(e)$ *is analogous.* $\square$

Theorem 5 *For any components P and Q in general the following properties of the composition* $P \otimes Q$ $(e \in Expression)$ *does NOT hold:*

$$P^{\mathsf{eout}}(e) \ \vee \ Q^{\mathsf{eout}}(e) \ \rightarrow \ (P \otimes Q)^{\mathsf{eout}}(e)$$
$$P_M^{\mathsf{eout}}(e) \ \vee \ Q_M^{\mathsf{eout}}(e) \ \rightarrow \ (P \otimes Q)_M^{\mathsf{eout}}(e)$$

Proof:
By the definition of eout *we have:*

$P^{\mathsf{eout}}(e) \ \vee \ Q^{\mathsf{eout}}(e)$

$\equiv$

$(\exists\, s_1 \in o_P : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s_1, t)) \ \vee \ (\exists\, s_2 \in o_Q : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s_2, t))$

$\equiv$

$\exists\, s \in (o_P \cup o_Q) : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s, t)$

$(P \otimes Q)^{\mathsf{eout}}(e)$

$\equiv$

$\exists\, s \in o_{P \otimes Q} : \ \exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s, t)$

By Equation 7 we have that $o_{P \otimes Q} \subseteq (o_P \cup o_Q)$, *i.e. in general we can have some stream* $s \in (o_P \cup o_Q)$ *for which* $\exists\, t \in \mathbb{N} : \ e \in \mathsf{ti}(s, t)$ *holds, but this stream does not necessary belongs to the set* $o_{P \otimes Q}$.

The proof for $P_M^{\mathsf{eout}}(e)$ *is analogous.* $\square$

Proposition 1 *For any components P and Q the composition* $P \otimes Q$ *has the following property:*

$$\neg P^{\mathsf{ine}}(m) \ \wedge \neg Q^{\mathsf{ine}}(m) \ \Rightarrow \ \neg \exists\, x \in l_{P \otimes Q} : \exists\, t \in \mathbb{N} : \ y \in \mathsf{ti}(x, t)$$

Proof:
By the definition of ine *we have:*

$$\neg P^{\mathsf{ine}}(m) \ \wedge \neg Q^{\mathsf{ine}}(m)$$

$$\equiv$$

$$\neg(\exists\, s \in i_P : \ \exists\, t \in \mathbb{N} : \ m \in \mathsf{ti}(s,t)) \ \wedge \ \neg(\exists\, s \in i_Q : \ \exists\, t \in \mathbb{N} : \ m \in \mathsf{ti}(s,t))$$

$$\equiv$$

$$\forall\, s \in i_P : \ \forall\, t \in \mathbb{N} : \ m \notin \mathsf{ti}(s,t) \ \wedge \ \forall\, s \in i_Q : \ \forall\, t \in \mathbb{N} : \ m \notin \mathsf{ti}(s,t)$$

$$\equiv$$

$$\forall\, s \in (i_P \cup i_Q) : \ \forall\, t \in \mathbb{N} : \ m \notin \mathsf{ti}(s,t)$$

According to the negation rules:

$$\neg\, \exists\, x \in l_{P \otimes Q} : \exists\, t \in \mathbb{N} : \ y \in \mathsf{ti}(x,t)$$

$$\equiv$$

$$\forall\, x \in l_{P \otimes Q} : \forall\, t \in \mathbb{N} : \ y \notin \mathsf{ti}(x,t)$$

By definition of the set of local streams and according to Equation 10 we have that $l_{P \otimes Q} \subseteq (i_P \cup i_Q)$ *and*

$$\forall\, s \in (i_P \cup i_Q) : \ \forall\, t \in \mathbb{N} : \ m \notin \mathsf{ti}(s,t)$$

$$\Rightarrow$$

$$\forall\, x \in l_{P \otimes Q} : \forall\, t \in \mathbb{N} : \ y \notin \mathsf{ti}(x,t)$$

□

Proposition 2 *For any two sets of streams* S_1 *and* S_2*, and for any secret* $m \in KS$ *the following relation holds:*

$$\forall\, A : i_A \subseteq S_1 \ \wedge \ m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m) \ \ \wedge$$

$$\forall\, A : i_A \subseteq S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m)$$

$$\Rightarrow$$

$$\forall\, A : i_A \subseteq S_1 \cup S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m)$$

Proof:
By the definition of ine *we have:*

$$\forall A : i_A \subseteq S_1 \;\wedge\; m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m) \;\wedge$$
$$\forall A : i_A \subseteq S_2 \;\wedge\; m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m)$$
$$\equiv$$
$$\forall A : i_A \subseteq S_1 \;\wedge\; m \notin KS_A \Rightarrow \neg(\exists\, s \in i_A : \;\exists\, t \in \mathbb{N} : \; m \in \mathsf{ti}(s,t)) \;\wedge$$
$$\forall A : i_A \subseteq S_2 \;\wedge\; m \notin KS_A \Rightarrow \neg(\exists\, s \in i_A : \;\exists\, t \in \mathbb{N} : \; m \in \mathsf{ti}(s,t))$$
$$\equiv$$
$$\forall A : i_A \subseteq S_1 \;\wedge\; m \notin KS_A \Rightarrow \forall\, s \in i_A : \;\forall\, t \in \mathbb{N} : \; m \notin \mathsf{ti}(s,t) \;\wedge$$
$$\forall A : i_A \subseteq S_2 \;\wedge\; m \notin KS_A \Rightarrow \forall\, s \in i_A : \;\forall\, t \in \mathbb{N} : \; m \notin \mathsf{ti}(s,t)$$

Because here the argumentation goes over all components A with $i_A \subseteq S_1$ and $i_A \subseteq S_2$, we can simplify this expression to

$$\forall\, s \in S_1 : \;\forall\, t \in \mathbb{N} : \; m \notin \mathsf{ti}(s,t) \;\wedge\; \forall\, s \in S_2 : \;\forall\, t \in \mathbb{N} : \; m \notin \mathsf{ti}(s,t)$$

which is equal to the expression

$$\forall\, s \in (S_1 \cup S_2) : \;\forall\, t \in \mathbb{N} : \; m \notin \mathsf{ti}(s,t)$$

This is a simplification of an expression

$$\forall A : i_A \subseteq (S_1 \cup S_2) \;\wedge\; m \notin KS_A \Rightarrow \forall\, s \in i_A : \;\forall\, t \in \mathbb{N} : \; m \notin \mathsf{ti}(s,t)$$

which corresponds to our goal

$$\forall A : i_A \subseteq S_1 \cup S_2 \;\wedge\; m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m)$$

□

Proposition 3 *For any two sets of streams S_1 and S_2, and for any secret $m \in KS$ the following relation holds:*

$$\exists A : i_A \subseteq S_1 \cup S_2 \;\wedge\; m \notin KS_A \;\wedge\; A^{\mathsf{ine}}(m) \Rightarrow$$
$$\exists A : i_A \subseteq S_1 \;\wedge\; m \notin KS_A \;\wedge\; A^{\mathsf{ine}}(m) \;\vee$$
$$\exists A : i_A \subseteq S_2 \;\wedge\; m \notin KS_A \;\wedge\; A^{\mathsf{ine}}(m)$$

Proof:
By the definition of ine *we have:*

$$\exists A : i_A \subseteq S_1 \cup S_2 \;\wedge\; m \notin KS_A \;\wedge\; A^{\mathsf{ine}}(m)$$
$$\equiv$$
$$\exists A : i_A \subseteq S_1 \cup S_2 \;\wedge\; m \notin KS_A \;\wedge\; (\exists\, s \in i_A : \;\exists\, t \in \mathbb{N} : \; m \in \mathsf{ti}(s,t))$$

Because here the argumentation goes only input streams of a component A with $i_A \subseteq S_1 \cup S_2$, we can simplify this expression to

$$\exists\, s \in (S_1 \cup S_2) : \;\forall\, t \in \mathbb{N} : \; m \notin \mathsf{ti}(s,t)$$

which is equal to the following expression

$$\exists\, s \in S_1 : \;\; \forall\, t \in \mathbb{N} : \;\; m \notin \mathit{ti}(s,t) \quad \vee \quad \exists\, s \in S_2 : \;\; \forall\, t \in \mathbb{N} : \;\; m \notin \mathit{ti}(s,t)$$

This is a simplification of an expression, which corresponds to our goal

$$\exists\, A : i_A \subseteq S_1 \;\wedge\; m \notin KS_A \;\wedge\; A^{\mathsf{ine}}(m) \;\vee$$
$$\exists\, A : i_A \subseteq S_2 \;\wedge\; m \notin KS_A \;\wedge\; A^{\mathsf{ine}}(m)$$

□

4.3 Knowledges of An Adversary

In addition to the sets of private keys and unguessable values of a component A we define the set of *local secrets* LS_A – the set of secrets which does not belong to the KS_A, but are transmitted via local channels of A or belongs to the local secrets of its subcomponents:

$$LS_A \stackrel{\mathsf{def}}{=}$$
$$\{m \in KS \mid m \notin KS_A \;\wedge\; \exists\, x \in l_A : \exists\, t \in \mathbb{N} : \;\; m \in \mathsf{ti}(x,t))\} \cup$$
$$\bigcup_{B \in \mathsf{subcomp}(A)} LS_B$$

If A is an elementary component and the set l_A of its local channels is empty, then also the set LS_A will be empty.

For a local secret m of a component A we denote by $t^A_{LS}(m)$ the first point in time at which m was transmitted via local channels:

$$t^A_{LS} \in LS_A \to \mathbb{N}$$

$$t^A_{LS}(m) = min(\{t \in \mathbb{N} \mid x \in l_A : \;\; m \in \mathsf{ti}(x,t)\} \cup \{t^B_{LS}(m) \mid B \in \mathsf{subcomp}(A)\})$$

An (*adversary*) component A knows a secret $m \in KS$, $m \notin KS_A$ (or some secret expression m, $m \in (\mathit{Expression} \setminus KS_A)^*$), if

- A may eventually get the secret m,
- m belongs to the set LS_A of its local secrets,
- A knows a one secret $\langle m \rangle$,
- A knows some list of expressions m_2 which is an concatenations of m and some list of expressions m_1,
- m is a concatenation of some secrets m_1 and m_2 ($m = m_1 \frown m_2$), and A knows both these secrets,
- A knows some secret key k^{-1} and the result of the encryption of the m with the corresponding public key,
- A knows some public key k and the result of the signature creation of the m with the corresponding private key,
- m is an encryption of some secret m_1 with a public key k, and A knows both m_1 and k,

- m is the result of the signature creation of the m_1 with the key k, and A knows both m_1 and k.

In the formal definition we need to distinguish two cases, represented by mutually recursive functions: m is a single secret or m some expression (or list), containing a secret – predicates $\mathsf{know}^A(k)$ and $\mathsf{knows}^A(k)$ respectively.

$$\mathsf{know}^A \in KS \setminus KS_A \to \mathbb{B}\mathrm{ool}$$
$$\mathsf{knows}^A \in (Expression \setminus KS_A)^* \to \mathbb{B}\mathrm{ool}$$

$$\mathsf{know}^A(m) \stackrel{\mathsf{def}}{=} A^{\mathsf{ine}}(m) \;\vee\; m \in LS_A$$

$$\begin{array}{l}
\mathsf{knows}^A(m) \stackrel{\mathsf{def}}{=} \\
(\exists\, m_1 : m = \langle m_1 \rangle \;\wedge\; \mathsf{know}^A(m_1)) \;\vee \\
(\exists\, m_1, m_2 : (m_2 = m \frown m_1 \;\vee\; m_2 = m_1 \frown m) \wedge \mathsf{knows}^A(m_2)) \;\vee \\
(\exists\, m_1, m_2 : m = m_1 \frown m_2 \;\wedge\; \mathsf{knows}^A(m_1) \;\wedge\; \mathsf{knows}^A(m_2)) \;\vee \\
(\exists\, k, k^{-1} : \mathsf{know}^A(k^{-1}) \;\wedge\; \mathsf{knows}^A(Enc(k, m))) \;\vee \\
(\exists\, k, k^{-1} : \mathsf{know}^A(k) \;\wedge\; \mathsf{knows}^A(Sign(k^{-1}, m))) \;\vee \\
(\exists\, k, m_1 : m = Enc(k, m_1) \;\wedge\; \mathsf{knows}^A(m_1) \;\wedge\; \mathsf{know}^A(k)) \;\vee \\
(\exists\, k, m_1 : m = Sign(k, m_1) \;\wedge\; \mathsf{knows}^A(m_1) \;\wedge\; \mathsf{know}^A(k))
\end{array}$$

For an adversary A who knows the secret m, we denote by $t^A_{\mathsf{know}}(m)$ ($t^A_{\mathsf{knows}}(m)$) the point in time from which A knows m.

$$t^A_{\mathsf{know}} \in KS \setminus KS_A \to \mathbb{N}^\infty$$
$$t^A_{\mathsf{knows}} \in (Expression \setminus KS_A)^* \to \mathbb{N}^\infty$$

$$\begin{array}{l}
t^A_{\mathsf{know}}(m) = \\
min\{\; \infty, \\
(\mathsf{if}\;\; A^{\mathsf{ine}}(m) \;\;\mathsf{then}\;\; min\{t \in \mathbb{N} : \; m \in \mathsf{ti}(s, t)\} \;\mathsf{else}\;\; \infty \;\mathsf{fi}\;), \\
(\mathsf{if}\;\; LS_A \neq \varnothing \\
\qquad \mathsf{then}\;\; t^A_{LS}(m) \;\mathsf{else}\;\; \infty \;\mathsf{fi}\;) \\
\}
\end{array}$$

$t^A_{\mathsf{knows}}(m) =$

$min\{\ \infty,$

$(\text{if}\ \ \exists\, m_1 : m = \langle m_1\rangle\ \wedge\ \mathsf{know}^A(m_1)\ \ \text{then}\ \ t^A_{\mathsf{know}}(m_1)\ \text{else}\ \ \infty\ \text{fi}\ ),$

$(\text{if}\ \exists\, m_1, m_2 : (m_2 = m \frown m_1\ \vee\ m_2 = m_1 \frown m) \wedge \mathsf{knows}^A(m_2)$

$\quad \text{then}\ \ t^A_{\mathsf{knows}}(m_2)\ \text{else}\ \ \infty\ \text{fi}\ ),$

$(\text{if}\ \ \exists\, m_1, m_2 : m = m_1 \frown m_2\ \wedge\ \mathsf{knows}^A(m_1)\ \wedge\ \mathsf{knows}^A(m_2)$

$\quad \text{then}\ \ max\{t^A_{\mathsf{knows}}(m_1), t^A_{\mathsf{knows}}(m_2)\}\ \text{else}\ \ \infty\ \text{fi}\ ),$

$(\text{if}\ \ \exists\, k, k^{-1} : \mathsf{know}^A(k^{-1})\ \wedge\ \mathsf{knows}^A(Enc(k, m))$

$\quad \text{then}\ \ max\{t^A_{\mathsf{know}}(k^{-1}), t^A_{\mathsf{knows}}(Enc(k, m))\}\ \text{else}\ \ \infty\ \text{fi}\ ),$

$(\text{if}\ \ \exists\, k, k^{-1} : \mathsf{know}^A(k)\ \wedge\ \mathsf{knows}^A(Sign(k^{-1}, m)))$

$\quad \text{then}\ \ \text{then}\ max\{t^A_{\mathsf{know}}(k), t^A_{\mathsf{knows}}(Sign(k^{-1}, m))\}\ \text{else}\ \ \infty\ \text{fi}\ ),$

$(\text{if}\ \ \exists\, k, m_1 : m = Enc(k, m_1)\ \wedge\ \mathsf{knows}^A(m_1)\ \wedge\ \mathsf{know}^A(k)$

$\quad \text{then}\ \ \text{then}\ max\{t^A_{\mathsf{know}}(k), t^A_{\mathsf{knows}}(m_1))\}\ \text{else}\ \ \infty\ \text{fi}\ ),$

$(\text{if}\ \ \exists\, k, m_1 : m = Sign(k, m_1)\ \wedge\ \mathsf{knows}^A(m_1)\ \wedge\ \mathsf{know}^A(k)$

$\quad \text{then}\ \ \text{then}\ max\{t^A_{\mathsf{know}}(k), t^A_{\mathsf{knows}}(m_1))\}\ \text{else}\ \ \infty\ \text{fi}\ )$

$\}$

Proposition 4 *If there exists an* adversary *component A_1 which knows a secret $m \in KS$ (or $m \in (Expression \setminus KS_A)^*$), then there exists an* adversary *component A_2 may eventually output this secret m:*

$$\exists\, A_1 :\ \ \mathit{know}^{A_1}(m)\ \ \Rightarrow\ \ \exists\, A_2 :\ \ {A_2}^{\mathit{eout}}(m)$$

$$\exists\, A_1 :\ \ \mathit{knows}^{A_1}(m)\ \ \Rightarrow\ \ \exists\, A_2 :\ \ {A_2}^{\mathit{eout}}(m)$$

Proof: *Assuming there exists an* adversary *component A_1 which knows a secret $m \in KS$. We can construct an* adversary *component A_2, which may eventually output this secret m, in the following way. We extend the semantics of the component A_1 by a channel mchannel of type KS and add to the body part of the specification by the corresponding formula:*

$$i_{A_2} = i_{A_1}\ \wedge$$

$$o_{A_2} = o_{A_1} \cup \{mchannel : KS\}\ \wedge$$

$$Body_{A_2} = (Body_{A_1}\ \ \wedge\ \ mchannel = \forall\, t :\ \ (t < t^{A_1}_{\mathit{know}}(m) \to \langle\rangle) \wedge (t \geq t^{A_1}_{\mathit{know}}(m) \to \langle m\rangle))$$

□

Axiom 1 *For any component C and for any secret $m \in KS$ (or expression $e \in Expression^*$), the following equations hold:*

$$\forall\, C :\ \ \forall\, m \in KS :\ \ C^{\mathit{eout}}(m)\ \equiv\ (m \in KS_C)\ \vee\ \mathit{know}^C(m)$$

$$\forall\, C :\ \ \forall\, e \in Expression^* :\ \ C^{\mathit{eout}}(e)\ \equiv\ (e \in {KS_C}^*)\ \vee\ \mathit{knows}^C(e)$$

□

Axiom 2 *For any component C and for an empty expression $\langle\rangle \in Expression^*$), the following equation holds:*

$$\forall\, C : \ \mathsf{knows}^C(\langle\rangle) = \mathsf{true}$$

$\square$

Proposition 5 *For any component C and for any secret $m \in KS$ the following equation holds:*

$$\forall\, C : \ \forall\, m \in KS : \ \mathsf{know}^C(m) = \mathsf{knows}^C(\langle m \rangle)$$

Proof: *Follows from the definition of predicate* knows. $\square$

Proposition 6 *If an* adversary *component A may eventually output a secret $m \in KS$ (or $m' \in (Expression \setminus KS_A)^*$), then this component A* knows *this secret m (m'):*

$$\forall\, A : \ A^{\mathsf{eout}}(m) \ \Rightarrow \ \mathsf{know}^A(m)$$
$$\forall\, A : \ A^{\mathsf{eout}}(m') \ \Rightarrow \ \mathsf{knows}^A(m')$$

Proof: *Follows from Axiom 1.* $\square$

Proposition 7 *If an* adversary *component A does not know a secret $m \in KS$, then this component A cannot eventually get this secret m:*

$$\forall\, A : \ \neg \mathsf{know}^A(m) \ \Rightarrow \ \neg A^{\mathsf{ine}}(m)$$

Proof: *Follows from the definition of* know. $\square$

Proposition 8 *If an* adversary *component A does not know a secret $m \in KS$ (or $m' \in (Expression \setminus KS_A)^*$), then this component A cannot eventually output this secret m:*

$$\forall\, A : \ \neg \mathsf{know}^A(m) \ \Rightarrow \ \neg A^{\mathsf{eout}}(m)$$
$$\forall\, A : \ \neg \mathsf{knows}^A(m) \ \Rightarrow \ \neg A^{\mathsf{eout}}(m)$$

Proof: *Follows from Axiom 1.* $\square$

Proposition 9 *If an* adversary *component A does not know a secret $m \in KS$ (or $m' \in (Expression \setminus KS_A)^*$), than the component P with $i_A \subseteq o_P$ cannot eventually output this secret:*

$$\forall A : i_A \subseteq o_P \wedge \neg \mathsf{know}^A(m) \;\Rightarrow\; \neg P^{\mathsf{eout}}(m)$$
$$\forall A : i_A \subseteq o_P \wedge \neg \mathsf{knows}^A(m) \;\Rightarrow\; \neg P^{\mathsf{eout}}(m)$$

Proof:
By Proposition 7 component A cannot eventually get the secret m: $\neg A^{\mathsf{ine}}(m)$, i.e.

$$\neg \exists s \in i_A : \; \exists t \in \mathbb{N} : \; m \in \mathsf{ti}(s,t)$$

Because we have here the quantification over all possible adversaries A with $i_A \subseteq o_P$, we can say that

$$\neg \exists s \in o_P : \; \exists t \in \mathbb{N} : \; m \in \mathsf{ti}(s,t)$$

which is exactly a negation of the predicate $P^{\mathsf{eout}}(m)$. □

Theorem 6 *For any components P and Q the composition $P \otimes Q$ has the following property ($m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$):*

$$\mathsf{know}^P(m) \;\Rightarrow\; \mathsf{know}^{P \otimes Q}(m)$$

Proof:
From the definition of know*:*

$$\mathsf{know}^P(m) \;\equiv\; P^{\mathsf{ine}}(m) \;\vee\; m \in LS_P$$

By Equation 6 we have that

$$i_{P \otimes Q} \subseteq (i_P \cup i_Q),$$

and, more exactly, by Equation 4 we have that

$$i_{P \otimes Q} = (i_P \cup i_Q) \setminus l_{P \otimes Q}$$

(1) If $P^{\mathsf{ine}}(m)$ holds:

$$\exists s \in i_P : \; \exists t \in \mathbb{N} : \; m \in \mathsf{ti}(s,t)$$

(1a) If $s \in i_{P \otimes Q}$ than we get

$$\exists s \in i_{P \otimes Q} : \; \exists t \in \mathbb{N} : \; m \in \mathsf{ti}(s,t)$$

which is definition of $\mathsf{know}^{P \otimes Q}(m)$.

(1b) Otherwise, if $s \notin i_{P\otimes Q}$, we have that $s \in l_{P\otimes Q}$, i.e. $m \in LS_{P\otimes Q}$ and $\mathsf{know}^{P\otimes Q}(m)$ holds by definition.

(2) If $m \in LS_P$, then by definition of LS we get $m \in LS_{P\otimes Q}$ and $\mathsf{know}^{P\otimes Q}(m)$ holds by definition. □

Theorem 7 *For any components P and Q the composition $P\otimes Q$ has the following property ($m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$):*

$$\mathsf{know}^{Q}(m) \;\Rightarrow\; \mathsf{know}^{P\otimes Q}(m)$$

Proof: *Analogous to the proof of Theorem 6.* □

Theorem 8 *For any components P and Q the composition $P\otimes Q$ has the following property ($m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$):*

$$\mathsf{know}^{P}(m) \;\vee\; \mathsf{know}^{Q}(m) \;\Rightarrow\; \mathsf{know}^{P\otimes Q}(m)$$

Proof: *Follows from Theorems 6 and 7.* □

Proposition 10 *For any components P and Q the composition $P\otimes Q$ has the following properties ($e \in KS^*$):*

$$\mathsf{knows}^{P}(\langle m\rangle) \;\Rightarrow\; \mathsf{knows}^{P\otimes Q}(\langle m\rangle) \qquad (1)$$
$$\mathsf{knows}^{Q}(\langle m\rangle) \;\Rightarrow\; \mathsf{knows}^{P\otimes Q}(\langle m\rangle) \qquad (2)$$

Proof:
From $\mathsf{knows}^{P}(\langle m\rangle)$ follows by Proposition 5 that $\mathsf{know}^{P}(m)$ holds. According to Theorem 6 we get that $\mathsf{know}^{P\otimes Q}(m)$ holds, which implies by Proposition 5 that $\mathsf{knows}^{P\otimes Q}(\langle m\rangle)$ holds.

From $\mathsf{knows}^{Q}(\langle m\rangle)$ follows by Proposition 5 that $\mathsf{know}^{Q}(m)$ holds. According to Theorem 7 we get that $\mathsf{know}^{P\otimes Q}(m)$ holds, which implies by Proposition 5 that $\mathsf{knows}^{P\otimes Q}(\langle m\rangle)$ holds. □

Theorem 9 *For any components P and Q the composition $P\otimes Q$ has the following property ($e \in KS^*$):*

$$\mathsf{knows}^{P}(e) \;\Rightarrow\; \mathsf{knows}^{P\otimes Q}(e)$$

Proof:
Let prove the first relation by induction over e:

Base case: $e = \langle\rangle$. According to Axiom 2: $\mathsf{knows}^{P\otimes Q}(\langle\rangle) = \mathsf{true}$.

Induction case: Assume, $\mathsf{knows}^P(e) \;\Rightarrow\; \mathsf{knows}^{P \otimes Q}(e)$ *holds. We need to prove that* $\mathsf{knows}^P(\langle m \rangle \frown e) \;\Rightarrow\; \mathsf{knows}^{P \otimes Q}(\langle m \rangle \frown e)$.

If the predicate $\mathsf{knows}^P(\langle m \rangle \frown e)$ *does not hold, the implication is simply true.*

Assuming, that $\mathsf{knows}^P(\langle m \rangle \frown e)$ *holds. From the definition of the predicate* knows *follows that* $\mathsf{knows}^P(\langle m \rangle)$ *and according to Proposition 10 we have* $\mathsf{knows}^{P \otimes Q}(\langle m \rangle)$.

Together with the induction assumption this implies by the definition of the predicate knows *that* $\mathsf{knows}^{P \otimes Q}(\langle m \rangle \frown e)$ *holds.* □

Theorem 10 *For any components* P *and* Q *the composition* $P \otimes Q$ *has the following property* ($e \in KS^*$)*:*

$$\mathsf{knows}^Q(e) \;\Rightarrow\; \mathsf{knows}^{P \otimes Q}(e)$$

Proof: *Analogous to the proof of Theorem 9.* □

Theorem 11 *For any components* P *and* Q *the composition* $P \otimes Q$ *has the following property* ($e \in KS^*$)*:*

$$\mathsf{knows}^P(e) \;\vee\; \mathsf{knows}^Q(e) \;\Rightarrow\; \mathsf{knows}^{P \otimes Q}(e)$$

Proof: *From Theorems 9 and 10.* □

Proposition 11 *For any components* P *and* Q *the following property of the composition* $P \otimes Q$ ($m \in KS$, $m \notin KS_P$ *and* $m \notin KS_Q$) *holds:*

$$\neg P^{\mathsf{ine}}(m) \;\wedge \neg Q^{\mathsf{ine}}(m) \;\wedge\; m \notin (LS_P \cup LS_Q) \;\Rightarrow\; m \notin LS_{P \otimes Q}$$

Proof:
By the definition of the local secrets set we have

$$\begin{aligned} LS_{P \otimes Q} = \quad & LS_P \;\cup\; LS_Q \;\cup \\ & \{y \in KS \mid y \notin (KS_P \cup KS_Q) \;\wedge\; \exists\, x \in l_{P \otimes Q} : \exists\, t \in \mathbb{N} : \; y \in \mathsf{ti}(x, t)\} \end{aligned}$$

The relation $m \notin (LS_P \cup LS_Q)$ *holds, and the relation (*m *cannot be transmitted via local channels of the composition* $P \otimes Q$*)*

$$m \notin \{y \in KS \mid y \notin (KS_P \cup KS_Q) \;\wedge\; \exists\, x \in l_{P \otimes Q} : \exists\, t \in \mathbb{N} : \; y \in \mathsf{ti}(x, t)\}$$

follows from Proposition 1. □

Theorem 12 *For any components P and Q the following properties of the composition $P \otimes Q$ ($m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$) hold:*

$$\neg \mathsf{know}^P(m) \ \wedge \ \neg \mathsf{know}^Q(m) \ \Rightarrow \ \neg \mathsf{know}^{P \otimes Q}(m) \qquad (1)$$
$$\mathsf{know}^{P \otimes Q}(m) \ \Rightarrow \ \mathsf{know}^P(m) \ \vee \ \mathsf{know}^Q(m) \qquad (2)$$

Proof:
(1) By the definition of the predicate know*:*

$$\neg \mathsf{know}^P(m) \ \wedge \ \neg \mathsf{know}^Q(m)$$
$$\equiv$$
$$\neg(P^{\mathsf{ine}}(m) \ \vee \ m \in LS_P) \ \wedge \ \neg(Q^{\mathsf{ine}}(m) \ \vee \ m \in LS_Q)$$
$$\equiv$$
$$\neg P^{\mathsf{ine}}(m) \ \wedge \ m \notin LS_P \ \wedge \ \neg Q^{\mathsf{ine}}(m) \ \wedge \ m \notin LS_Q$$
$$\equiv$$
$$\neg P^{\mathsf{ine}}(m) \ \wedge \ \neg Q^{\mathsf{ine}}(m) \ \wedge \ m \notin (LS_P \cup LS_Q)$$

$$\neg \mathsf{know}^{P \otimes Q}(m)$$
$$\equiv$$
$$\neg((P \otimes Q)^{\mathsf{ine}}(m) \ \vee \ m \in LS_{P \otimes Q})$$
$$\equiv$$
$$\neg(P \otimes Q)^{\mathsf{ine}}(m) \ \wedge \ m \notin LS_{P \otimes Q})$$

If follows from $\neg P^{\mathsf{ine}}(m) \ \wedge \neg Q^{\mathsf{ine}}(m)$ by Theorem 3 that $\neg(P \otimes Q)^{\mathsf{ine}}(m)$ holds.

The second conjunct, $m \notin LS_{P \otimes Q}$, can be proven according according to Proposition 11 from the expression $\neg P^{\mathsf{ine}}(m) \ \wedge \ \neg Q^{\mathsf{ine}}(m) \ \wedge \ m \notin (LS_P \cup LS_Q)$. □

Theorem 13 *For any components P and Q the following properties of the composition $P \otimes Q$ ($e \in KS^*$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$) hold:*

$$\neg \mathsf{knows}^P(\langle m \rangle) \ \wedge \ \neg \mathsf{knows}^Q(\langle m \rangle) \ \Rightarrow \ \neg \mathsf{knows}^{P \otimes Q}(\langle m \rangle) \qquad (1)$$
$$\mathsf{knows}^{P \otimes Q}(\langle m \rangle) \ \Rightarrow \ \mathsf{knows}^P(\langle m \rangle) \ \vee \ \mathsf{knows}^Q(\langle m \rangle) \qquad (2)$$

Proof:
The first relation can be proven by Proposition 5 and Theorem 12(1):

$\neg\mathsf{knows}^P(\langle m\rangle) \ \wedge\ \neg\mathsf{knows}^Q(\langle m\rangle)$

$\equiv$

$\neg\mathsf{know}^P(m) \ \wedge\ \neg\mathsf{know}^Q(m)$

$\Rightarrow$

$\neg\mathsf{know}^{P\otimes Q}(m)$

$\equiv$

$\neg\mathsf{knows}^{P\otimes Q}(\langle m\rangle)$

The second relation can be proven by Proposition 5 and Theorem 12(2):

$\mathsf{knows}^{P\otimes Q}(\langle m\rangle)$

$\equiv$

$\mathsf{know}^{P\otimes Q}(m)$

$\Rightarrow$

$\mathsf{know}^P(m) \ \vee\ \mathsf{know}^Q(m)$

$\equiv$

$\mathsf{knows}^P(\langle m\rangle) \ \vee\ \mathsf{knows}^Q(\langle m\rangle)$

□

Theorem 14 *For any components P and Q in general the following properties of the composition $P \otimes Q$ ($e \in KS^{\underline{\omega}}$, $\neg\exists\, m \in KS : e = \langle m\rangle$, $m \notin KS_P$ and $m \notin KS_Q$) does NOT hold:*

$$\neg\mathsf{knows}^P(e) \ \wedge\ \neg\mathsf{knows}^Q(e) \ \Rightarrow\ \neg\mathsf{knows}^{P\otimes Q}(e) \qquad (1)$$

$$\mathsf{knows}^{P\otimes Q}(e) \ \Rightarrow\ \mathsf{knows}^P(e) \ \vee\ \mathsf{knows}^Q(e) \qquad (2)$$

Proof:
Let discuss counter-examples to the relations above. From the definition of knows*, if $\nexists\, m \in KS : e = \langle m\rangle$, we always have the case, that to know m, we need to know two corresponding expressions – then we can "derivate" $\mathsf{knows}^A(m)$. E.g., if $\exists\, m_1, m_2 : e = m_1 \frown m_2$, we can derivate $\mathsf{knows}^A(m)$ from $\mathsf{knows}^A(m_1)$ and $\mathsf{knows}^A(m_2)$. Thus, we can represent all these cases by*

$$e = SomeRelation(e_1, e_2) \ \Rightarrow\ (\mathsf{knows}^A(e_1) \ \wedge\ \mathsf{knows}^A(e_2)) = \mathsf{knows}^A(e_1)$$

where A is some component with corresponding KS_A, i.e. P, Q, or $P \otimes Q$.

Assuming the situation where $\mathsf{knows}^P(e_1)$, but $\neg\mathsf{knows}^P(e_2)$, and $\mathsf{knows}^Q(e_2)$, but $\neg\mathsf{knows}^Q(e_1)$.

Thus, we have here that $\neg\mathsf{knows}^P(e)$ and $\neg\mathsf{knows}^Q(e)$.

The relation $\neg\mathsf{knows}^P(e) \wedge \neg\mathsf{knows}^Q(e)$ holds, and the relation $\mathsf{knows}^P(e) \vee \mathsf{knows}^Q(e)$ does not hold.

By Theorem 6 we get that the relation $\mathsf{knows}^{P\otimes Q}(e_1)$ *holds, and by Theorem 7 we get that the relation* $\mathsf{knows}^{P\otimes Q}(e_2)$ *holds. Thus, we can derivate that* $\mathsf{knows}^{P\otimes Q}(e)$ *holds also, and this disproves the relations (1) and (2).* □

Proposition 12 *For any two sets of streams* S_1 *and* S_2*, and for any secret* $m \in KS$ *the following relation holds:*

$$\forall A : i_A \subseteq S_1 \ \wedge \ m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m) \ \wedge$$
$$\forall A : i_A \subseteq S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$
$$\Rightarrow$$
$$\forall A : i_A \subseteq S_1 \cup S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

Proof:
By the definition of know *we have:*

$$\forall A : i_A \subseteq S_1 \ \wedge \ m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m) \ \wedge$$
$$\forall A : i_A \subseteq S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$
$$\equiv$$
$$\forall A : i_A \subseteq S_1 \ \wedge \ m \notin KS_A \Rightarrow \neg (A^{\mathsf{ine}}(m) \ \vee \ m \in LS_A) \ \wedge$$
$$\forall A : i_A \subseteq S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg (A^{\mathsf{ine}}(m) \ \vee \ m \in LS_A)$$
$$\equiv$$
$$\forall A : i_A \subseteq S_1 \ \wedge \ m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m) \ \wedge \ m \notin LS_A \ \wedge$$
$$\forall A : i_A \subseteq S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m) \ \wedge \ m \notin LS_A$$

By the definition of the set of local secrets, it is independent of the set of input streams of the component. Thus, according to this definition and to Proposition 2, we get

$$\forall A : i_A \subseteq S_1 \cup S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m) \ \wedge \ m \notin LS_A$$
$$\equiv$$
$$\forall A : i_A \subseteq S_1 \cup S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg A^{\mathsf{ine}}(m) \ \vee \ m \in LS_A)$$
$$\equiv$$
$$\forall A : i_A \subseteq S_1 \cup S_2 \ \wedge \ m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

□

Proposition 13 *For any two sets of streams* S_1 *and* S_2*, and for any secret* $m \in KS$ *the following relation holds:*

$$\exists A : i_A \subseteq S_1 \cup S_2 \ \wedge \ m \notin KS_A \ \wedge \ \mathsf{know}^A(m) \Rightarrow$$
$$\exists A : i_A \subseteq S_1 \ \wedge \ m \notin KS_A \ \wedge \ \mathsf{know}^A(m) \ \vee$$
$$\exists A : i_A \subseteq S_2 \ \wedge \ m \notin KS_A \ \wedge \ \mathsf{know}^A(m)$$

Proof:
By the definition of know *we have:*

$$\exists\, A : i_A \subseteq S_1 \cup S_2 \;\wedge\; m \notin KS_A \;\wedge\; \mathsf{know}^A(m)$$
$$\equiv$$
$$\exists\, A : i_A \subseteq S_1 \cup S_2 \;\wedge\; m \notin KS_A \;\wedge\; (A^{\mathsf{ine}}(m) \;\vee\; m \in LS_A)$$

By the definition of the set of local secrets, it is independent of the set of input streams of the component. Thus, according to this definition and to Proposition 3, we get

$$\exists\, A : i_A \subseteq S_1 \;\wedge\; m \notin KS_A \Rightarrow (A^{\mathsf{ine}}(m) \;\vee\; m \in LS_A) \;\vee$$
$$\exists\, A : i_A \subseteq S_2 \;\wedge\; m \notin KS_A \Rightarrow (A^{\mathsf{ine}}(m) \;\vee\; m \in LS_A)$$
$$\equiv$$
$$\exists\, A : i_A \subseteq S_1 \;\wedge\; m \notin KS_A \Rightarrow \mathsf{know}^A(m) \;\vee$$
$$\exists\, A : i_A \subseteq S_2 \;\wedge\; m \notin KS_A \Rightarrow \mathsf{know}^A(m)$$

□

4.4 Preserving The Secrecy

We say that a component P *leaks a secret* $m \in KS$ (denoted by $P^{\mathsf{leak}}(m)$) if there exists an *adversary* component A with $i_A \subseteq o_P$ and $m \notin KS_A$ such that the composition $P \otimes A$ may eventually output m:

$$P^{\mathsf{leak}}(m) \stackrel{\mathsf{def}}{=} \exists\, A : i_A \subseteq o_P \;\wedge\; m \notin KS_A \;\wedge\; (P \otimes A)^{\mathsf{eout}}(m)$$

Otherwise we say that P *preserves the secrecy of* m (denoted by $P^{\mathsf{secr}}(m)$):

$$P^{\mathsf{secr}}(m) \stackrel{\mathsf{def}}{=} \forall\, A : i_A \subseteq o_P \;\wedge\; m \notin KS_A \;\Rightarrow\; \neg(P \otimes A)^{\mathsf{eout}}(m)$$

With other words $P^{\mathsf{leak}}(m)$ means, that for some $t \in \mathbb{N}$ $m \in \mathsf{ti}(x, t)$, where x is either an output channel of P, which "goes outside" of our system, or an output channel of some component A which "goes outside" of the composition $P \otimes A$ (a component A communicates with P directly).

Proposition 14 *A component P leaks a secret m iff there exists an adversary component A with $i_A \subseteq o_P$ and $m \notin KS_A$ such that which knows a secret m:*

$$P^{\mathsf{leak}}(m) \Leftrightarrow$$
$$\exists\, A : i_A \subseteq o_P \wedge m \notin KS_A \wedge \mathsf{know}^A(m)$$

Proof:
Let prove the both directions of the equation.
(1)*:*

$$P^{\mathsf{leak}}(m) \Rightarrow \exists\, A : i_A \subseteq o_P \wedge m \notin KS_A \wedge \mathsf{know}^A(m)$$

Applying the definition of $P^{\mathsf{leak}}(m)$:

$$\exists\, A : i_A \subseteq o_P \;\wedge\; m \notin KS_A \;\wedge\; (P \otimes A)^{\mathsf{eout}}(m) \;\Rightarrow$$
$$\exists\, A : i_A \subseteq o_P \;\wedge\; m \notin KS_A \;\wedge\; \mathsf{know}^A(m)$$

The case when the adversary A with properties $i_A \subseteq o_P$, $m \notin KS_A$ and $(P \otimes A)^{\mathsf{eout}}(m)$ does not exists, is trivial – the implication holds.

Assuming now, that such an adversary A exists. Then we need to prove, that from the property $(P \otimes A)^{\mathsf{eout}}(m)$ follows that $\mathsf{know}^{A'}(m)$ holds (A' is not necessary equal to A, it can be some refinement of the component A).

By the definition of $(P \otimes A)^{\mathsf{eout}}(m)$: there exists some time interval t of an output stream $s \in o_{P \otimes A}$ of the composition $P \otimes A$ which contains the expression m. Here we have two cases:

- *$s \in o_A$, $s \notin o_P$ – this case is trivial. $m \notin KS_A$ means that to output m, A need first of all to receive this message or extract it from another received messages, i.e. A need to know the secret m (according to our definition). Thus, $\mathsf{know}^A(m)$ holds and we can define $A' = A$.*

- *$s \in o_P$, $s \notin o_A$.*
  We define A' as an interface refinement of the component A: $i_{A'} = i_A \cup \{s\}$, thus, ${A'}^{\mathsf{ine}}(m)$ holds and this implies that $\mathsf{know}^{A'}(m)$ holds also.

(2)*:*

$$P^{\mathsf{leak}}(m) \Leftarrow \exists\, A : i_A \subseteq o_P \wedge m \notin KS_A \wedge \mathsf{know}^A(m)$$

The case when the adversary A with properties $i_A \subseteq o_P$, $m \notin KS_A$ and $\mathsf{knows}^A(m)$ does not exists, is trivial – the implication holds.

Assuming now, that such an adversary A exists. By Proposition 4 we have

$$\exists\, A' :\; {A'}^{\mathsf{eout}}(m),$$

where by Proposition 4

$$i_{A'} = i_A.$$

Thus, $i_{A'} \subseteq o_P$ and $m \notin KS_{A'}$ hold, $(P \otimes A')^{\mathsf{eout}}(m)$ holds also by the definition. □

Proposition 15 *A component P preserves the secrecy of m iff there does not exist an adversary component A with $i_A \subseteq o_P$ and $m \notin KS_A$ such that which knows a secret m:*

$$P^{\mathsf{secr}}(m) \Leftrightarrow$$
$$\forall\, A : i_A \subseteq o_P \wedge m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

Proof:
Let prove the both directions of the equation.
(1)*:*

$$P^{\mathsf{secr}}(m) \Rightarrow \forall\, A : i_A \subseteq o_P \wedge m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

By the definition of $P^{\mathsf{secr}}(m)$

$$\forall\, A : (i_A \subseteq o_P \wedge m \notin KS_A \;\Rightarrow\; \neg (P \otimes A)^{\mathsf{eout}}(m)) \;\Rightarrow$$
$$\forall\, A : i_A \subseteq o_P \wedge m \notin KS_A \;\Rightarrow\; \neg \mathsf{know}^A(m)$$

To have more clear proof structure, we rename the first quantifier to A'*:*

$$\forall\, A' : (i_{A'} \subseteq o_P \wedge m \notin KS_{A'} \;\Rightarrow\; \neg (P \otimes A')^{\mathsf{eout}}(m)) \;\Rightarrow$$
$$\forall\, A : i_A \subseteq o_P \wedge m \notin KS_A \;\Rightarrow\; \neg \mathsf{know}^A(m)$$

Assuming there exists an adversary A*, s.t.* $i_A \subseteq o_P$, $m \notin KS_A$ *and* $\mathsf{know}^A(m)$*. By Proposition 4 we have that* $\exists\, A_2 :\; {A_2}^{\mathsf{eout}}(m)$*, where the equality* $i_{A_2} = i_A$ *holds. Thus,* $i_{A_2} \subseteq o_P$ *and* $m \notin KS_{A_2}$ *hold,* $(P \otimes A_2)^{\mathsf{eout}}(m)$ *holds also by the definition. This is a contradiction.*
(2)*:*

$$P^{\mathsf{secr}}(m) \Leftarrow \forall\, A : i_A \subseteq o_P \wedge m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

We have, that for any adversary A*,* A *cannot know the secret* m*. By Proposition 9 we get,* P *cannot eventually output the secret* m*, and by Propositions 8 we get also, that for any adversary* A*,* A *cannot eventually output the secret* m*.*

This implies (by Theorem 3 that the composition $P \otimes A$ *cannot eventually output the secret* m *(for any adversary* A *with* $i_A \subseteq o_P$ *and* $m \notin KS_A$*):*

$$\forall\, A : i_A \subseteq o_P \;\wedge\; m \notin KS_A \;\Rightarrow\; \neg (P \otimes A)^{\mathsf{eout}}(m)$$

which is exactly the definition of $P^{\mathsf{secr}}(m)$*.* □

Proposition 16 *For any components* P *and* A*, such that* $i_A = o_P \;\wedge\; i_P = o_A$ *the composition* $P \otimes A$ *cannot output any expression* $E \in CExp$ *(i.e.* $(P \otimes A)^{\mathsf{eout}}(E)$*), because the set of output stream of the composition* $P \otimes A$*. We call such a composed systems a* closed *one.* □

Given a relation $C \subseteq O_P \times I_P$ from the set of output streams of a component P to the set of input streams of P, we say that P *leaks* m *assuming* C for $m \in KS$ (denoted by $P^{\mathsf{leak}}_C(m)$), if there exists a component A (in our system) with $m \notin S_A \cup K_A$ that fulfills C and such that $P \otimes A$ may eventually output m.

$$P^{\mathsf{leak}}_C(m) \overset{\mathsf{def}}{=} \exists\, A : i_A \subseteq o_P \;\wedge\; m \notin S_A \cup K_A \;\wedge$$
$$(P \otimes A)^{\mathsf{eout}}(m) \;\wedge\; ([\![\, A \,]\!] \rightarrow [\![\, C \,]\!])$$

Otherwise P *preserves the secrecy of* m *assuming* C.

Theorem 15 *If P_1 preserves the secrecy of m and $P_1 \rightsquigarrow P_2$ then P_2 preserves the secrecy of m:*

$$({P_1}^{\mathsf{secr}}(m) \ \wedge \ (P_1 \rightsquigarrow P_2)) \Rightarrow {P_2}^{\mathsf{secr}}(m)$$

Proof:
According to the idea of the refinement-based verification [1], we can represent the secrecy property ${P_1}^{\mathsf{secr}}(m)$ as a detached specification P_0.

The refinement relation $P_0 \rightsquigarrow P_1$ holds, and we get the refinement hierarchy $P_0 \rightsquigarrow P_1 \rightsquigarrow P_2$.

Thus, we can say, that $P_0 \rightsquigarrow P_2$, i.e. that the secrecy property holds for P_2: ${P_2}^{\mathsf{secr}}(m)$. □

Theorem 16 *If P_1 preserves the secrecy of m assuming C (for any $C \subseteq O_{P_1} \times I_{P_1}$) and $P_1 \rightsquigarrow P_2$ then P_2 preserves the secrecy of m assuming C:*

$$({P_1}^{\mathsf{secr}}_C(m) \ \wedge \ (P_1 \rightsquigarrow P_2)) \Rightarrow {P_2}^{\mathsf{secr}}_C(m)$$

Proof: *Analog to Theorem 15.* □

To argue about knowledges of a component in a definite time, we also introduce a predicate $\mathsf{got}_t(s)$ which returns for a stream s the set of messages, which occurs in the stream until the time t, and

$$\mathsf{got}_t \in M^{\underline{\infty}} \Rightarrow \mathbb{P}(M)$$
$$\mathsf{got}_0(s) = \mathsf{set}(\mathsf{ti}(s, 0))$$
$$\mathsf{got}_{t+1}(s) = \mathsf{got}_t(s) \cup \mathsf{set}(\mathsf{ti}(s, t+1))$$

On this base we can define the function $\mathsf{knows}^A_t \subseteq (\mathit{Expression} \setminus KS_A)$ which returns for the component the set of secrets, known until the time t analog to the function t_{knows}.

Theorem 17 *For any components P and Q the composition $P \otimes Q$ has the following properties ($m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$):*

$$P^{\mathsf{secr}}(m) \ \wedge \ Q^{\mathsf{secr}}(m) \ \rightarrow \ (P \otimes Q)^{\mathsf{secr}}(m) \qquad (1)$$
$$(P \otimes Q)^{\mathsf{leak}}(m) \ \rightarrow \ P^{\mathsf{leak}}(m) \ \vee \ Q^{\mathsf{leak}}(m) \qquad (2)$$

Proof:
(1) $P^{\mathsf{secr}}(m) \ \wedge \ Q^{\mathsf{secr}}(m) \ \rightarrow \ (P \otimes Q)^{\mathsf{secr}}(m)$
By Proposition 15:

$$P^{\mathsf{secr}}(m) \ \wedge \ Q^{\mathsf{secr}}(m)$$
$$\equiv$$
$$\forall\, A : i_A \subseteq o_P \wedge m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m) \ \wedge$$
$$\forall\, A : i_A \subseteq o_Q \wedge m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

By Proposition 12 we get

$$\forall A : i_A \subseteq o_P \cup o_Q \;\wedge\; m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

By Proposition 7 about output streams of a composite component we have that

$$o_{P \otimes Q} \subseteq (o_P \;\cup\; o_Q).$$

This implies that

$$\forall A : i_A \subseteq o_{P \otimes Q} \;\wedge\; m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

holds, which is exactly the definition of $(P \otimes Q)^{\mathsf{secr}}(m)$.

(2) $(P \otimes Q)^{\mathsf{leak}}(m) \;\rightarrow\; P^{\mathsf{leak}}(m) \;\vee\; Q^{\mathsf{leak}}(m)$
By Proposition 14:

$$\begin{array}{l} (P \otimes Q)^{\mathsf{leak}}(m) \\ \equiv \\ \exists\, A : i_A \subseteq o_{(P \otimes Q)} \wedge m \notin KS_A \wedge \mathsf{know}^A(m) \end{array}$$

By Proposition 7 about output streams of a composite component we have that

$$o_{P \otimes Q} \subseteq (o_P \;\cup\; o_Q).$$

This implies that the relation

$$\exists\, A : i_A \subseteq (o_P \cup o_Q) \wedge m \notin KS_A \wedge \mathsf{know}^A(m)$$

holds. By Proposition 13 we can derivate

$$\begin{array}{l} \exists\, A : i_A \subseteq o_P \wedge m \notin KS_A \wedge \mathsf{know}^A(m) \;\;\vee \\ \exists\, A : i_A \subseteq o_Q \wedge m \notin KS_A \wedge \mathsf{know}^A(m) \end{array}$$

which is exactly the definition of $P^{\mathsf{leak}}(m) \;\vee\; Q^{\mathsf{leak}}(m)$. □

Theorem 18 *For any components* P *and* Q *in general the following properties of the composition* $P \otimes Q$ *(*$m \in KS$*,* $m \notin KS_P$ *and* $m \notin KS_Q$*) does NOT hold:*

$$(P \otimes Q)^{\mathsf{secr}}(m) \;\Rightarrow\; P^{\mathsf{secr}}(m) \;\wedge\; Q^{\mathsf{secr}}(m) \qquad (1)$$

$$P^{\mathsf{leak}}(m) \;\vee\; Q^{\mathsf{leak}}(m) \;\Rightarrow\; (P \otimes Q)^{\mathsf{leak}}(m) \qquad (2)$$

Proof:
(1) $(P \otimes Q)^{\mathsf{secr}}(m) \;\Rightarrow\; P^{\mathsf{secr}}(m) \;\wedge\; Q^{\mathsf{secr}}(m)$
By Proposition 15:

$(P \otimes Q)^{\mathsf{secr}}(m)$

$\equiv$

$\forall\, A : i_A \subseteq o_{P \otimes Q} \wedge m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$

This expression does not exclude, that some of components P or Q can output the message m via some stream x, because this situation can be "covered" by the composition – if $x \in l_{P \otimes Q}$. Thus, the expression

$$\forall\, A : i_A \subseteq o_P \wedge m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m) \ \wedge \forall\, A : i_A \subseteq o_Q \wedge m \notin KS_A \Rightarrow \neg \mathsf{know}^A(m)$$

does not hold in general, which implies that we cannot derivate in general the expression $P^{\mathsf{secr}}(m) \ \wedge \ Q^{\mathsf{secr}}(m)$.

(2) $P^{\mathsf{leak}}(m) \ \vee \ Q^{\mathsf{leak}}(m) \ \Rightarrow \ (P \otimes Q)^{\mathsf{leak}}(m)$
By Proposition 14:

$P^{\mathsf{leak}}(m) \ \vee \ Q^{\mathsf{leak}}(m)$

$\equiv$

$\exists\, A : i_A \subseteq o_P \wedge m \notin KS_A \wedge \mathsf{know}^A(m) \quad \vee \quad \exists\, A : i_A \subseteq o_Q \wedge m \notin KS_A \wedge \mathsf{know}^A(m)$

This expression does not exclude, that some of components P or Q can output the message m such a stream x, which belongs to the set of local streams of the composition $P \otimes Q$, i.e. $x \in l_{P \otimes Q}$. Thus, the component $P \otimes Q$ in general will be not necessary output the message m, and the expression

$$\exists\, A : i_A \subseteq o_{P \otimes Q} \wedge m \notin KS_A \wedge \mathsf{know}^A(m)$$

does not hold in general, which implies that we cannot derivate in general the expression $(P \otimes Q)^{\mathsf{leak}}(m)$. □

5 TLS Protocol

To demonstrate usability of our approach, we specify a variant of the handshake protocol of TLS[1] [3] (note that this is not the variant of TLS in common use). The goal of the TLS protocol is to let a client send a secret over an untrusted communication link to a server in a way that provides secrecy and server authentication, by using symmetric session keys.

The protocol has two participants: *Client* and *Server* that are connected by an Internet connection. The value *secretD* which is exchanged encrypted in the last message of the protocol is required to remain secret. The value $genKey \in Keys$ is a session key, which is symmetric (i.e. $genKey^{-1} = genKey$) and is generated by the server. This implies that $\mathsf{knows}^A(genKey)$ holds if and only if $\mathsf{knows}^A(genKey^{-1})$ holds.

To specify this protocol in Focus we will use the following auxiliary data types: *Obj* of participants names, *State* of participant states, *Event* of message sending events (e.g. an abort message, an acknowledgment etc.), and *InitMessage* representing the event that initiates the protocol by the client.

$$\mathsf{type}\ Obj = \{C,\ S\}$$

$$\mathsf{type}\ State = \{initS, waitS\}$$

$$\mathsf{type}\ Event = \{event\}$$

$$\mathsf{type}\ InitMessage = im(ungValue \in Secret,$$
$$key \in Keys,\ msg \in Expression)$$

The protocol assumes that there is a secure (wrt. integrity) way for the client to obtain the public key *CAKey* of the certification authority, and for the server to obtain a certificate $Sign(CAKey^{-1}, \langle S, SKey \rangle)$ signed by the certification authority that contains its name and public key. An adversary may also have access to *CAKey*, $Sign(CAKey^{-1}, \langle S, SKey \rangle)$ and $Sign(CAKey^{-1}, \langle Z, ZKey \rangle)$ for an arbitrary process Z.

5.1 The Handshake Protocol

Client initiates the protocol by sending the message that contains an unguessable value $N \in Secret$, its the public key K_C, and a sequence $\langle C, CKey \rangle$ of its name and its public key signed by its secret key K_C^{-1}.

Server checks whether the received public key matches to the second element of the signed sequence. If that is the case, it returns to the *Client* the received unguessable value N, an encryption of a sequence $\langle genKey, N \rangle$ (signed by its secret key $SKey^{-1}$) using the received public key, and a sequence $\langle S, SKey \rangle$ (of its name and its public key) signed using the secret key $CAKey^{-1}$ of the certification authority *CA*.

Client checks whether the certificate is actually for S and the correct N is returned. If that is the case, it sends the secret value *secretD* encrypted with the received session key *genKey* to the *Server* .

If any of the checks fail, the respective protocol participant stops the execution of the protocol by sending an abort signal.

[1] TLS (Transport Layer Security) is the successor of the Internet security protocol SSL (Secure Sockets Layer).

Client($CKey, CKey^{-1} \in Keys$) — timed

in $abortS : Event;\ resp : Expression$

out $init : InitMessage, xchd : Expression;$
$abortC : Event$

---

asm $\mathsf{msg}_2(resp) \wedge \mathsf{msg}_1(abortS)$
$\forall\, E \in Expression :$
$Decr(CKey^{-1}, Enc(CKey, E)) = E$

- - -

gar

$\mathsf{ti}(init, 0) =$
$\langle im(N, CKey, Sign(CKey^{-1}, \langle C, CKey\rangle))\rangle$
$\mathsf{ti}(xchd, 0) = \langle\rangle$
$\mathsf{ti}(abortC, 0) = \langle\rangle$

$\forall\, t \in \mathbb{N} :\ \mathsf{ti}(init, t+1) = \langle\rangle$

$\forall\, t \in \mathbb{N} :\ \mathsf{ti}(abortS, t) \neq \langle\rangle \rightarrow$
$\mathsf{ti}(xchd, t+1) = \langle\rangle\ \wedge \mathsf{ti}(abortC, t+1) = \langle\rangle$

$\forall\, t \in \mathbb{N} :$
$\mathsf{ti}(abortS, t) = \langle\rangle \rightarrow$
$(\mathsf{ti}(resp, t) = \langle\rangle \rightarrow$
$\mathsf{ti}(xchd, t) = \langle\rangle\ \wedge\ \mathsf{ti}(abortC, t+1) = \langle\rangle)$
$\wedge$
$(\mathsf{ti}(resp, t) \neq \langle\rangle \rightarrow$
$\mathsf{ft}.Ext(CAKey, resp^t_{\mathsf{trd}}) = S\ \wedge$
$\mathsf{snd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}),$
$Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) = N\ \rightarrow$
$\mathsf{ti}(abortC, t+1) = \langle\rangle\ \wedge$
$\mathsf{ti}(xchd, t+1) =$
$Enc(\mathsf{ft}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}),$
$Decr(CKey^{-1}, resp^t_{\mathsf{snd}})),$
$secretD)$
$\wedge$
$\mathsf{ft}.Ext(CAKey, resp^t_{\mathsf{snd}}) \neq S\ \vee$
$\mathsf{snd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{snd}}),$
$Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) \neq N\ \rightarrow$
$\mathsf{ti}(abortC, t+1) = \langle event\rangle\ \wedge$
$\mathsf{ti}(xchd, t+1) = \langle\rangle$

Server(SKey, SKey^{-1} $\in$ *Keys*) — timed

in $init : InitMessage;\ abortC : Event;$
$xchd : Expression$

out $resp : Expression;\ abortS : Event$

---

local $stateS \in StateS$

---

init $stateS = initS$

---

asm $\mathsf{msg}_1(init) \ \wedge\ \mathsf{msg}_1(xchd)$

$\forall\, E \in Expression :$

$Decr(SKey^{-1}, Enc(SKey, E)) = E$

---

gar

$\mathsf{ti}(resp, 0) = \langle\rangle \ \wedge\ \mathsf{ti}(abortS, 0) = \langle\rangle$

$\forall\, t \in \mathbb{N} :$

$\mathsf{ti}(abortC, t) \neq \langle\rangle \rightarrow$

$stateS' = initS \ \wedge\ \mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge$

$\mathsf{ti}(abortS, t+1) = \langle\rangle$

$\wedge$

$\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = initS \ \rightarrow$

$(\mathsf{ti}(init, t) = \langle\rangle \rightarrow$

$\mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ stateS' = initS)$

$\wedge$

$(\mathsf{ti}(init, t) \neq \langle\rangle \rightarrow$

$\mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) \neq key(init^t_{\mathsf{ft}})$

$\rightarrow$

$\mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ stateS' = initS \ \wedge$

$\mathsf{ti}(abortS, t+1) = \langle event\rangle$

$\wedge$

$\mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) = key(init^t_{\mathsf{ft}})$

$\rightarrow$

$\mathsf{ti}(resp, t+1) = \langle e0, e1, e2\rangle \ \wedge$

$stateS' = waitS \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle))$

$\wedge$

$\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = waitS \ \rightarrow$

$\mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ stateS' = waitS$

where

$e0, e1, e2$ so that

$e0 = ungValue(init^t_{\mathsf{ft}}),$

$e1 = Enc(key(init^t_{\mathsf{ft}}),$

$Sign(SKey^{-1}, \langle genKey, ungValue(init^t_{\mathsf{ft}})\rangle)),$

$e2 = Sign(CAKey^{-1}, \langle S, SKey\rangle)$

CA$(\mathrm{CAKey}^{-1}, \mathrm{CAKey} \in \mathit{Keys})$ — timed

out $a_C : \mathit{Keys};\ a_A, a_S : \mathit{Expression}$

asm

$\forall\, E \in \mathit{Expression} :$

$\quad \mathit{Decr}(\mathit{CAKey}^{-1}, \mathit{Enc}(\mathit{CAKey}, E)) = E$

$\quad \mathit{Ext}(\mathit{CAKey}^{-1}, \mathit{Sign}(\mathit{CAKey}, E)) = E$

gar

$\mathsf{ti}(a_C, 0) = \langle \mathit{CAKey} \rangle$

$\mathsf{ti}(a_A, 0) = \langle \mathit{CAKey}, \mathit{Sign}(\mathit{CAKey}^{-1}, \langle S, \mathit{SKey} \rangle),$
$\qquad \mathit{Sign}(\mathit{CAKey}^{-1}, \langle Z, \mathit{ZKey} \rangle) \rangle$

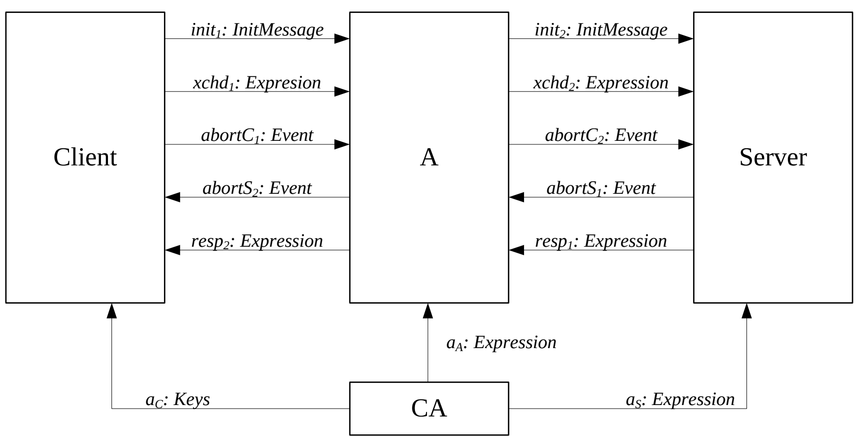
$\mathsf{ti}(a_S, 0) = \langle \mathit{Sign}(\mathit{CAKey}^{-1}, \langle S, \mathit{SKey} \rangle) \rangle$

$\forall\, t \in \mathbb{N} : \quad \mathsf{ti}(a_C, t+1) = \langle \rangle$

$\forall\, t \in \mathbb{N} : \quad \mathsf{ti}(a_A, t+1) = \langle \rangle$

$\forall\, t \in \mathbb{N} : \quad \mathsf{ti}(a_S, t+1) = \langle \rangle$

System$(\mathrm{SKey}^{-1}, \mathrm{SKey}, \mathrm{CKey}^{-1}, \mathrm{CKey}, \mathrm{CAKey}^{-1}, \mathrm{CAKey}, \in \mathit{Keys})$ — glass-box



5.2 Security Analysis

In this section, we use our approach to demonstrate a security flaw in the TLS variant introduced above, and how to correct it.

Theorem 3:
Let $P = \mathit{Client} \otimes \mathit{Server} \otimes \mathit{CA}$. P does not preserve the secrecy of m, where $m \in \mathit{KS}$: $P^{\mathsf{leak}}(m)$.
Proof: According to the specification of the *Client* component, we need to consider $m = \mathit{secretD}$. To show that $P^{\mathsf{leak}}(m)$, we need to find an adversary component A with $I_A \subseteq O_P$ such that $\mathsf{knows}^A(m)$ holds with regards to the composition, and m does not belong to the set of private keys of A or to the set of unguessable values of A:

$$\exists\, A : I_A \subseteq O_P \;\wedge\; m \notin \mathit{KS}_A \;\wedge\; \mathsf{knows}^A(m)$$

According to Prop. 2, this would imply that the predicate $P^{\mathsf{leak}}(m)$ holds.

Consider the Focus specification of the component A specified below. If we trace its knowledge base as its evolves in interaction with the protocol components, we get the following:

$$\begin{array}{ll} t = 0: & \mathsf{knows}^A(\mathit{CAKey}) \\ t = 1: & \mathsf{knows}^A(\mathit{Sign}(\mathit{CAKey}^{-1}, \langle S, \mathit{SKey}\rangle)) \\ & \mathsf{knows}^A(\mathit{SKey}) \\ & \mathsf{knows}^A(\mathit{Sign}(\mathit{SKey}^{-1}, \langle \mathit{genKey}, N\rangle)) \\ & \mathsf{knows}^A(\mathit{genKey}) \\ & \mathsf{knows}^A(\mathit{genKey}^{-1}) \\ t = 2: & \mathsf{knows}^A(\mathit{Enc}(\mathit{genKey}, \mathit{secretD})) \\ & \mathsf{knows}^A(\mathit{secretD}) \end{array}$$

Let us discuss the computations more precisely, step by step:

Initially, $t = 0$:
Client:
$\mathsf{ti}(\mathit{init}_1, 0) = \langle \mathit{im}(N, \mathit{CKey}, \mathit{Sign}(\mathit{CKey}^{-1}, \langle C, \mathit{CKey}\rangle))\rangle$
$\mathsf{ti}(\mathit{xchd}_1, 0) = \langle\rangle$, $\mathsf{ti}(\mathit{abortC}_1, 0) = \langle\rangle$
Server:
$\mathsf{ti}(\mathit{resp}_1, 0) = \langle\rangle$, $\mathsf{ti}(\mathit{abortS}_1, 0) = \langle\rangle$
A:
$\mathsf{ti}(\mathit{init}_2, 0) = \langle \mathit{im}(N, \mathit{AKey}, \mathit{Sign}(\mathit{AKey}^{-1}, \langle C, \mathit{AKey}\rangle))\rangle$
$\mathsf{ti}(\mathit{xchd}_2, 0) = \langle\rangle$, $\mathsf{ti}(\mathit{abortC}_2, 0) = \langle\rangle$
$\mathsf{ti}(\mathit{resp}_2, 0) = \langle\rangle$, $\mathsf{ti}(\mathit{abortS}_2, 0) = \langle\rangle$

$t = 1$:
Client:

$\mathsf{ti}(init_1, 1) = \langle\rangle$, $\mathsf{ti}(xchd_1, 1) = \langle\rangle$, $\mathsf{ti}(abortC_1, 1) = \langle\rangle$
Server:
$\mathsf{ti}(resp_1, 1) = \langle N, Enc(AKey, Sign(SKey^{-1}, \langle genKey, N\rangle)), Sign(CAKey^{-1}, \langle S, SKey\rangle)\rangle$,
$\mathsf{ti}(abortS_1, 1) = \langle\rangle$
because we have $\mathsf{ti}(init_2, 0) \neq \langle\rangle$ and

$$\begin{aligned}
&\mathsf{snd}.Ext(\langle key((init_2)^0_{\mathsf{ft}}), msg((init_2)^0_{\mathsf{ft}})\rangle) = \\
&\textit{according value of } \mathsf{ti}(init_2, 0) \\
&\mathsf{snd}.Ext(\langle AKey, Sign(AKey^{-1}, \langle C, AKey\rangle)\rangle) = \\
&\textit{according the relation between functions Ext and Sign} \\
&\mathsf{snd}.\langle C, CKey\rangle = \\
&\textit{by the definition of } \mathsf{snd}. \\
&AKey = \\
&\textit{according value of } \mathsf{ti}(init_2, 0) \\
&key((init_2)^0_{\mathsf{ft}})
\end{aligned}$$

and

$$\begin{aligned}
&Enc(key((init_2)^0_{\mathsf{ft}}), Sign(SKeySectret, \langle genKey, ungValue((init_2)^0_{\mathsf{ft}})\rangle)) = \\
&\textit{according value of } \mathsf{ti}(init_2, 0) \\
&Enc(AKey, Sign(SKeySectret, \langle genKey, N\rangle))
\end{aligned}$$

A:
$\mathsf{ti}(init_2, 1) = \langle\rangle$, $\mathsf{ti}(xchd_2, 1) = \langle\rangle$, $\mathsf{ti}(abortC_2, 1) = \langle\rangle$, $\mathsf{ti}(abortS_2, 1) = \langle\rangle$
$\mathsf{ti}(resp_2, 1) = \langle N, Enc(CKey, Sign(SKey^{-1}, \langle genKey, N\rangle)), Sign(CAKey^{-1}, \langle S, SKey\rangle)\rangle$

$t = 2$:
Client:
$\mathsf{ti}(init_1, 2) = \langle\rangle$, $\mathsf{ti}(abortC_1, 2) = \langle\rangle$
$\mathsf{ti}(xchd_1, 2) = Enc(genKey, secretD)$, because we have $\mathsf{ti}(resp, 1) \neq \langle\rangle$ and

$$\begin{aligned}
&Ext(CAKey, (resp_2)^1_{\mathsf{trd}}) = \\
&\textit{according value of } \mathsf{ti}(resp_2, 1) \\
&Ext(CAKey, Sign(CAKey^{-1}, \langle S, SKey\rangle)) = \\
&\textit{according the relation between functions Ext and Sign} \\
&\langle S, SKey\rangle
\end{aligned}$$

$$\mathsf{ft}.Ext(CAKey, (resp_2)^1_{\mathsf{trd}}) = S$$

$$\mathsf{snd}.Ext(CAKey, (resp_2)^1_{\mathsf{trd}}) = SKey$$

$Ext(\mathsf{snd}.Ext(CAKey, (resp_2)^1_{\mathsf{trd}}), Decr(CKey^{-1}, (resp_2)^1_{\mathsf{snd}})) =$

according value of $\mathsf{ti}(resp_2, 1)$

$Ext(SKey, Decr(CKey^{-1}, Enc(CKey, Sign(SKey^{-1}, \langle genKey, N\rangle)))) =$

according the relation between functions Decr and Enc

$Ext(SKey, Sign(SKey^{-1}, \langle genKey, N\rangle)) =$

according the relation between functions Ext and Sign

$\langle genKey, N\rangle$

$Enc(\mathsf{ft}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}})), secretD) =$

$Enc(genKey, secretD)$

Server:
$\mathsf{ti}(resp_1, 2) = \langle\rangle$,
$\mathsf{ti}(abortS_1, 2) = \langle\rangle$
A:
$\mathsf{ti}(init_2, 2) =$, $\mathsf{ti}(abortC_2, 2) = \langle\rangle$, $\mathsf{ti}(resp_2, 2) = \langle\rangle$, $\mathsf{ti}(abortS_2, 2) = \langle\rangle$
$\mathsf{ti}(xchd_2, 2) = Enc(genKey, secretD)$

□

A(AKey, AKey^{-1} $\in$ *Keys*) — timed

in	$abortC_1, abortS_1 : Event;\ xchd_1 : XS;$ $resp_1 : Expression;\ init_1 : InitMessage$
out	$abortC_2, abortS_2 : Event;\ xchd_2 : XS;$ $resp_2 : Expression;\ init_2 : InitMessage$

local $keyCP \in Keys;$

init $keyCP = AKey;$

asm $\mathsf{msg}_2(resp_1) \ \wedge\ \mathsf{msg}_1(abortS_1) \ \wedge$
$\mathsf{msg}_2(init_1) \ \wedge\ \mathsf{msg}_1(xchd_1)$

gar

$\forall\, t \in \mathbb{N} :\ \mathsf{ti}(abortC_2, t) = \mathsf{ti}(abortC_1, t)$

$\forall\, t \in \mathbb{N} :\ \mathsf{ti}(abortS_2, t) = \mathsf{ti}(abortS_1, t)$

$$\forall\, t \in \mathbb{N} : \quad \mathsf{ti}(init_1, t) = \langle\rangle \ \rightarrow \mathsf{ti}(init_2, t) = \langle\rangle$$

$$\forall\, t \in \mathbb{N} : \quad \mathsf{ti}(resp_1, t) = \langle\rangle \ \rightarrow \mathsf{ti}(resp_2, t) = \langle\rangle$$

$$\forall\, t \in \mathbb{N} : \quad \mathsf{ti}(xchd_2, t) = \mathsf{ti}(xchd_1, t)$$

$$\forall\, t \in \mathbb{N} : \quad \mathsf{ti}(init_1, t) \neq \langle\rangle \ \rightarrow keyCP' = key((init_1)^t_{\mathsf{ft}}) \ \wedge \ \mathsf{ti}(init_2, t) = \langle im(ungValue((init_1)^t_{\mathsf{ft}}), AKey, Sign(AKey^{-1}, \langle C, AKey\rangle))\rangle$$

$$\forall\, t \in \mathbb{N} : \quad \mathsf{ti}(resp_1, t) \neq \langle\rangle \ \rightarrow \mathsf{ti}(resp_2, t) = \langle (resp_1)^t_{\mathsf{ft}}, Enc(keyCP, Decr(AKey^{-1}, (resp_1)^t_{\mathsf{snd}})), (resp_1)^t_{\mathsf{trd}}\rangle$$

To fix this security weakness, we need to change the protocol: the client must find out the situation, where an adversary try to get the secret data. Thus, we need to correct the specification of the server in such a way that the client will know with which public key the data was encrypted at the server, and this information must be received by the client without any possible changes by the adversary. The only part of the messages from the server which cannot be changed by the adversary is the result of the signature creation – the adversary does not know the secret key $SKey^{-1}$ and cannot modify the signature or create a new one with modified content. Therefore, we add the public key received by the server to the content $\langle genKey, N\rangle$ of the signature. If there is not attack, this will be $CKey$, in the attack scenario explained above, it would be $AKey$. Accordingly, in the FOCUS specification of the *Server*, we change the definition of $e1$ to the following one:

$$
\begin{aligned}
&Enc(key(init^t_{\mathsf{ft}}),\\
&\quad Sign(SKey^{-1},\\
&\qquad \langle genKey, ungValue(init^t_{\mathsf{ft}}), key(init^t_{\mathsf{ft}})\rangle))
\end{aligned}
$$

Also, correspondingly we add a new conjunct to the condition for the correct data receipt in the specification of the client:

$$
\begin{aligned}
&\mathsf{trd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}),\\
&\qquad\qquad Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) = CKey
\end{aligned}
$$

Client(CKey, CKey^{-1} $\in$ *Keys*) — timed

in $abortS : Event;\ resp : Expression$

out $init : InitMessage, xchd : XS;\ abortC : Event$

---

asm $\mathsf{msg}_2(resp) \ \wedge\ \mathsf{msg}_1(abortS)$
$\quad \forall\, E \in Expression:\ Decr(CKey^{-1}, Enc(CKey, E)) = E$

- - - - - - - - - - - - - - - - - - - -

gar

$$
\begin{aligned}
&\mathsf{ti}(init, 0) = \langle im(N, CKey, Sign(CKey^{-1}, \langle C, CKey\rangle))\rangle\\
&\mathsf{ti}(xchd, 0) = \langle\rangle\\
&\mathsf{ti}(abortC, 0) = \langle\rangle\\[1em]
&\forall\, t \in \mathbb{N}:\quad \mathsf{ti}(init, t+1) = \langle\rangle\\[1em]
&\forall\, t \in \mathbb{N}:\ \mathsf{ti}(abortS, t) \neq \langle\rangle \rightarrow \mathsf{ti}(xchd, t+1) = \langle\rangle \ \wedge\ \mathsf{ti}(abortC, t+1) = \langle\rangle\\[1em]
&\forall\, t \in \mathbb{N}:\\
&\quad \mathsf{ti}(abortS, t) = \langle\rangle \rightarrow\\
&\qquad (\mathsf{ti}(resp, t) = \langle\rangle \rightarrow \mathsf{ti}(xchd, t) = \langle\rangle \ \wedge\ \mathsf{ti}(abortC, t+1) = \langle\rangle)\\
&\qquad \wedge\\
&\qquad (\mathsf{ti}(resp, t) \neq \langle\rangle \rightarrow\\
&\qquad\qquad \mathsf{ft}.Ext(CAKey, resp^t_{\mathsf{trd}}) = S\ \wedge\\
&\qquad\qquad \mathsf{snd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) = N\ \wedge\\
&\qquad\qquad \mathsf{trd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) = CKey\ \rightarrow\\
&\qquad\qquad\quad \mathsf{ti}(abortC, t+1) = \langle\rangle\ \wedge\\
&\qquad\qquad\quad \mathsf{ti}(xchd, t+1) =\\
&\qquad\qquad\qquad Enc(\mathsf{ft}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}})),\\
&\qquad\qquad\qquad\quad secretD)\\
&\qquad\qquad \wedge\\
&\qquad\qquad \mathsf{ft}.Ext(CAKey, resp^t_{\mathsf{snd}}) \neq S\ \vee\\
&\qquad\qquad \mathsf{snd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{snd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) \neq N\ \rightarrow\\
&\qquad\qquad\quad \mathsf{ti}(abortC, t+1) = \langle event\rangle\ \wedge\ \mathsf{ti}(xchd, t+1) = \langle\rangle
\end{aligned}
$$

Server($SKey, SKey^{-1} \in Keys$) — **timed**

in $init : InitMessage;\ abortC : Event;\ xchd : XS$

out $resp : Expression;\ abortS : Event$

---

local $stateS \in StateS$

init $stateS = initS$

asm $\mathsf{msg}_1(init) \wedge \mathsf{msg}_1(xchd)$
$\forall E \in Expression:\ Decr(SKey^{-1}, Enc(SKey, E)) = E$

gar
$\mathsf{ti}(resp, 0) = \langle\rangle \wedge \mathsf{ti}(abortS, 0) = \langle\rangle$

$$
\begin{array}{l}
\forall t \in \mathbb{N}: \\
\quad \mathsf{ti}(abortC, t) \neq \langle\rangle \rightarrow \\
\quad\quad stateS' = initS \wedge \mathsf{ti}(resp, t+1) = \langle\rangle \wedge \mathsf{ti}(abortS, t+1) = \langle\rangle \\
\quad \wedge \\
\quad \mathsf{ti}(abortC, t) = \langle\rangle \wedge stateS = initS \rightarrow \\
\quad\quad (\mathsf{ti}(init, t) = \langle\rangle \rightarrow \mathsf{ti}(resp, t+1) = \langle\rangle \wedge stateS' = initS) \\
\quad\quad \wedge \\
\quad\quad (\mathsf{ti}(init, t) \neq \langle\rangle \rightarrow \\
\quad\quad\quad \mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) \neq key(init^t_{\mathsf{ft}}) \rightarrow \\
\quad\quad\quad\quad \mathsf{ti}(resp, t+1) = \langle\rangle \wedge stateS' = initS \wedge \mathsf{ti}(abortS, t+1) = \langle event\rangle \\
\quad\quad\quad \wedge \\
\quad\quad\quad \mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) = key(init^t_{\mathsf{ft}}) \rightarrow \\
\quad\quad\quad\quad \mathsf{ti}(resp, t+1) = \langle e0, e1, e2\rangle \wedge stateS' = waitS \wedge \mathsf{ti}(abortS, t+1) = \langle\rangle)) \\
\quad \wedge \\
\quad \mathsf{ti}(abortC, t) = \langle\rangle \wedge stateS = waitS \rightarrow \\
\quad\quad \mathsf{ti}(resp, t+1) = \langle\rangle \wedge stateS' = waitS
\end{array}
$$

where
$e0, e1, e2$ so that
$e0 = ungValue(init^t_{\mathsf{ft}})$,
$e1 = Enc(key(init^t_{\mathsf{ft}}), Sign(SKey^{-1}, \langle genKey, ungValue(init^t_{\mathsf{ft}}), key(init^t_{\mathsf{ft}})\rangle))$,
$e2 = Sign(CAKey^{-1}, \langle S, SKey\rangle)$

Now, if we trace the knowledge base of the adversary A considered above, the secret is not leaked:

$$
\begin{array}{ll}
t = 0: & \mathsf{knows}^A(CAKey),\ \mathsf{knows}^A(CKey) \\
t = 1: & \mathsf{knows}^A(Sign(CAKey^{-1}, \langle S, SKey\rangle)) \\
 & \mathsf{knows}^A(SKey) \\
 & \mathsf{knows}^A(Sign(SKey^{-1}, \langle genKey, N, AKey\rangle)) \\
 & \mathsf{knows}^A(genKey) \\
 & \mathsf{knows}^A(genKey^{-1}) \\
t = 2: & --
\end{array}
$$

The transmission will be aborted by the client.

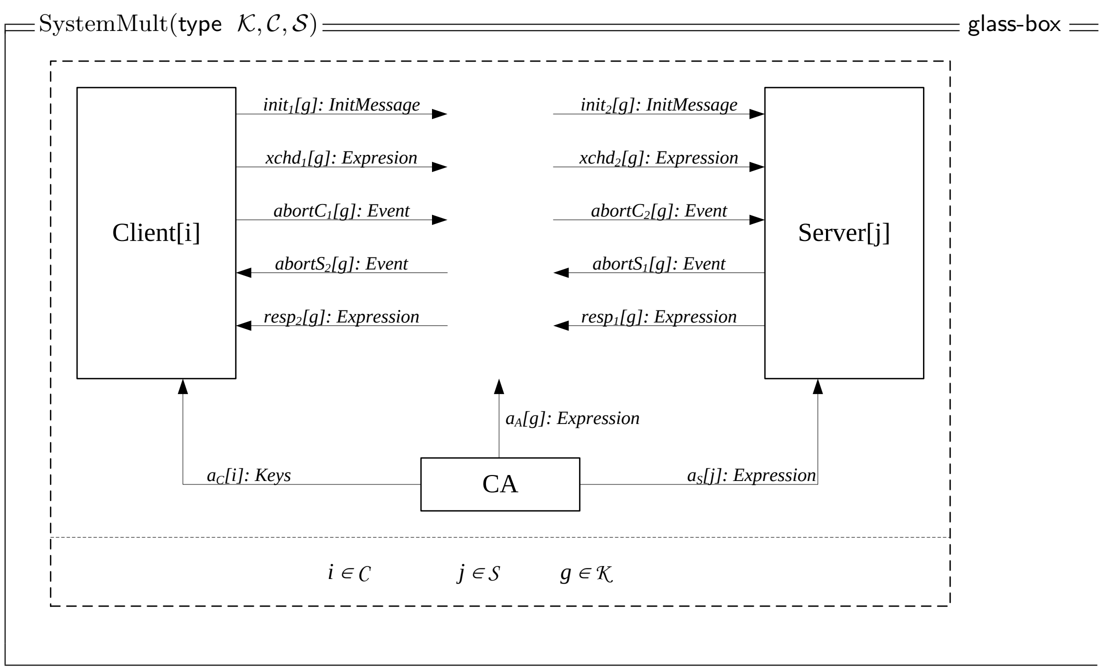
Using the formal approach explained above, one can also go further and prove that not only the attack described above is not possible anymore, but more generally there is no other attack by the kind of Dolev-Yao attacker considered here, which would get access to the secret.

Please note that here we actually do not need to argue about the input streams *abortC1* and *abortC2* of the component A, because these streams are of type *Event*, which has no relation with the type *Expression*.

5.3 Extension

We can also extend the fixed version of the specifications, e.g. by allowing multiple parallel sessions. To specify the situation where there are multiple parallel sessions, instead of just considering one session key *genKey* and the associated client C and server S, we consider a set of session keys $\mathcal{K}$, a set of clients $\mathcal{C}$, and a set of servers $\mathcal{S}$ (which may each be infinite), together with two functions $k_C : \mathcal{K} \to \mathcal{C}$ and $k_S : \mathcal{K} \to \mathcal{S}$ which determine which client and server are involved in a given session, which is represented by the session key (which is unique to the session, where the actual key generation is left implicit here).

The corresponding specification of the system in Focus will be defined using sheaves of channels and specification replication: see the specification *SystemMult* below.



6 Secure Channels

We sketch how one can formally develop a secure communication channel based on the crypto protocol verification approach explained in the previous section.

The components *ChC* and *ChS* are specified on the base of the fixed specifications of the simple client and server components (see Section 5.2). Here we are not interested in the detailed functionality of the components *ExternalClient* and *ExternalServer*, we just consider abstractions of two components where the component *ExternalClient* sends some data to the component *ExternalServer*.

If the *ExternalClient* receives the message d at the time unit t, there is no communication problem, and it sends messages only from the second time unit after t, then the *ExternalServer* gets this data at the time unit $t+2+\mathit{delay}$, where *delay* is a communication delay dependent on the communication medium, and the two time units delay arises from using the secure channels.[2] The *CA* component is the same as in Section 5.1.

To argue about the properties of the components which represent the secure channels we extend the definition of the predicate $\mathsf{knows}^A(m)$.

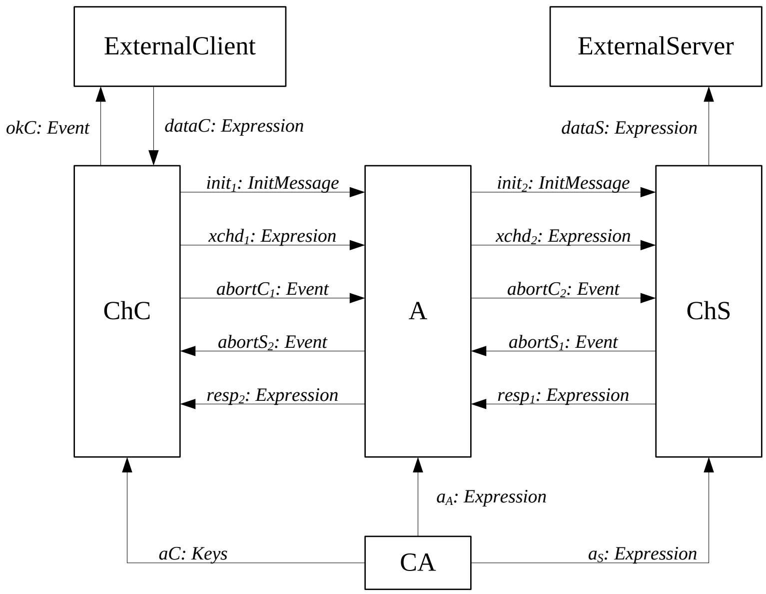
We say that an *adversary* component A knows a secret $m \in KS$, $m \notin KS_A$ via the set M of channels (denoted by $\mathsf{knows}^A_M(m)$), if we restrict the set of input streams of A to some its subset M, $M \subseteq I_A$.

Using the predicates $\mathsf{knows}^A(m)$ and $\mathsf{knows}^A_M(m)$ we can describe the knowledge data base of the component A, and, moreover, we can describe this for every time interval $t \in \mathbb{N}$.

We can then specify a system with secure channel components (without multiplication) in Focus as a composed component *ChSystem*.

[2] We get the two time units delay, because we model the secure channels as strong causal components to avoid anomalies such as the Brock-Ackerman anomaly. If one prefers to work with weak causality, the overall delay will be equal to the communication delay, but then the composition may not in general be well-defined.

$\mathrm{ChSystem}(\mathrm{SKey}^{-1}, \mathrm{SKey}, \mathrm{CKey}^{-1}, \mathrm{CKey}, \mathrm{CAKey}^{-1}, \mathrm{CAKey}, \in \mathit{Keys}) = \textsf{glass-box}$



ChC(CKey, $\text{CKey}^{-1} \in Keys$) — timed

in $\quad abortS : Event;\ dataC, resp : Expression$

out $\quad init : InitMessage, xchd : XS;\ abortC, okC : Event$

---

local $\quad stC \in StateS;\ \ buffer \in Expression^*;\ gkey \in Keys$

init $\quad stC = initS;\ \ buffer = \langle\rangle$

asm $\quad \mathsf{msg}_2(resp) \ \wedge\ \mathsf{msg}_1(abortS) \ \wedge\ \mathsf{msg}_1(dataC)$

$\quad \forall E \in Expression:\ \ Decr(CKey^{-1}, Enc(CKey, E)) = E$

gar

$\mathsf{ti}(init, 0) = \langle im(N, CKey, Sign(CKey^{-1}, \langle C, CKey\rangle))\rangle$

$\mathsf{ti}(xchd, 0) = \langle\rangle$

$\mathsf{ti}(abortC, 0) = \langle\rangle$

$\mathsf{ti}(okC, 0) = \langle\rangle$

$\forall t \in \mathbb{N}: \quad \mathsf{ti}(init, t+1) = \langle\rangle$

$\forall t \in \mathbb{N}:$

$\quad \mathsf{ti}(abortS, t) \neq \langle\rangle \rightarrow$

$\quad\quad \mathsf{ti}(xchd, t+1) = \langle\rangle \ \wedge\ \mathsf{ti}(abortC, t+1) = \langle\rangle \ \wedge\ stC' = initS \ \wedge\ \mathsf{ti}(okC, 0) = \langle\rangle$

$\forall t \in \mathbb{N}:$

$\quad \mathsf{ti}(abortS, t) = \langle\rangle \ \wedge\ stC = initS \rightarrow$

$\quad\quad \mathsf{ti}(okC, 0) = \langle\rangle$

$\quad\quad \wedge$

$\quad\quad (\mathsf{ti}(resp, t) = \langle\rangle \rightarrow \ \wedge\ \mathsf{ti}(abortC, t+1) = \langle\rangle \ \wedge\ stC' = initS)$

$\quad\quad \wedge$

$\quad\quad (\mathsf{ti}(resp, t) \neq \langle\rangle \rightarrow$

$\quad\quad\quad \mathsf{ft}.Ext(CAKey, resp^t_{\mathsf{trd}}) = S \ \wedge$

$\quad\quad\quad \mathsf{snd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) = N \ \wedge$

$\quad\quad\quad \mathsf{trd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) = CKey \ \rightarrow$

$\quad\quad\quad\quad \mathsf{ti}(abortC, t+1) = \langle\rangle \ \wedge\ \mathsf{ti}(xchd, t+1) = \langle\rangle \ \wedge\ stC' = waitS$

$\quad\quad\quad\quad \wedge\ gkey' = \mathsf{ft}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{trd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}}))$

$\quad\quad\quad \wedge$

$\quad\quad\quad \mathsf{ft}.Ext(CAKey, resp^t_{\mathsf{snd}}) \neq S \ \vee$

$\quad\quad\quad \mathsf{snd}.Ext(\mathsf{snd}.Ext(CAKey, resp^t_{\mathsf{snd}}), Decr(CKey^{-1}, resp^t_{\mathsf{snd}})) \neq N \ \rightarrow$

$\quad\quad\quad\quad \mathsf{ti}(abortC, t+1) = \langle event\rangle \ \wedge\ \mathsf{ti}(xchd, t+1) = \langle\rangle \ \wedge\ stC' = initS$

$\forall t \in \mathbb{N}:$

$\quad \mathsf{ti}(abortS, t) = \langle\rangle \ \wedge\ stC = waitS \rightarrow$

$\quad\quad (buffer = \langle\rangle \rightarrow$

$\quad\quad\quad (\mathsf{ti}(dataC, t) = \langle\rangle \rightarrow$

$\quad\quad\quad\quad \mathsf{ti}(xchd, t) = \langle\rangle \ \wedge\ buffer' = \langle\rangle \ \wedge\ \mathsf{ti}(okC, 0) = \langle\rangle)$

$\quad\quad\quad \wedge$

$\quad\quad\quad (\mathsf{ti}(dataC, t) \neq \langle\rangle \rightarrow$

$\quad\quad\quad\quad \mathsf{ti}(xchd, t+1) = \langle Enc(gkey, dataC^t_{\mathsf{ft}})\rangle \ \wedge$

$\quad\quad\quad\quad buffer' = \langle\rangle \ \wedge\ \mathsf{ti}(okC, 0) = \langle event\rangle))$

$\quad\quad \wedge$

$\quad\quad (buffer \neq \langle\rangle \rightarrow$

$\quad\quad\quad \mathsf{ti}(xchd, t+1) = \langle Enc(gkey, \mathsf{ft}.buffer)\rangle \ \wedge$

$\quad\quad\quad buffer' = \mathsf{rt}.buffer \frown \mathsf{ti}(dataC, t) \ \wedge\ \mathsf{ti}(okC, 0) = \langle event\rangle)$

$\forall t \in \mathbb{N}:$

$\quad stC = waitS \rightarrow gkey' = gkey$

$\forall t \in \mathbb{N}:$

$\quad (\mathsf{ti}(abortS, t) \neq \langle\rangle \ \vee\ stC = initS) \rightarrow buffer' = buffer \frown \mathsf{ti}(dataC, t)$

$ChS(SKey, SKey^{-1} \in Keys)$ — timed

in $init : InitMessage;\ abortC : Event;\ xchd : XS$

out $dataS, resp : Expression;\ abortS : Event$

---

local $stateS \in StateS$

init $stateS = initS$

asm $\mathsf{msg}_1(init) \wedge \mathsf{msg}_1(xchd)$
$\forall E \in Expression:\ Decr(SKey^{-1}, Enc(SKey, E)) = E$

gar
$\mathsf{ti}(resp, 0) = \langle\rangle \wedge \mathsf{ti}(abortS, 0) = \langle\rangle \wedge \mathsf{ti}(dataS, 0) = \langle\rangle$

$$
\begin{array}{l}
\forall t \in \mathbb{N}: \\
\quad \mathsf{ti}(abortC, t) \neq \langle\rangle \rightarrow \\
\quad\quad stateS' = initS \wedge \mathsf{ti}(resp, t+1) = \langle\rangle \wedge \mathsf{ti}(abortS, t+1) = \langle\rangle \wedge \mathsf{ti}(dataS, t+1) = \langle\rangle \\
\quad \wedge \\
\quad \mathsf{ti}(abortC, t) = \langle\rangle \wedge stateS = initS \rightarrow \\
\quad\quad \mathsf{ti}(dataS, t+1) = \langle\rangle \\
\quad\quad \wedge \\
\quad\quad (\mathsf{ti}(init, t) = \langle\rangle \rightarrow \mathsf{ti}(resp, t+1) = \langle\rangle \wedge stateS' = initS) \\
\quad\quad \wedge \\
\quad\quad (\mathsf{ti}(init, t) \neq \langle\rangle \rightarrow \\
\quad\quad\quad \mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) \neq key(init^t_{\mathsf{ft}}) \rightarrow \\
\quad\quad\quad\quad \mathsf{ti}(resp, t+1) = \langle\rangle \wedge stateS' = initS \wedge \mathsf{ti}(abortS, t+1) = \langle event\rangle \\
\quad\quad\quad \wedge \\
\quad\quad\quad \mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) = key(init^t_{\mathsf{ft}}) \rightarrow \\
\quad\quad\quad\quad \mathsf{ti}(resp, t+1) = \langle e0, e1, e2\rangle \wedge stateS' = waitS \wedge \mathsf{ti}(abortS, t+1) = \langle\rangle)) \\
\quad \wedge \\
\quad \mathsf{ti}(abortC, t) = \langle\rangle \wedge stateS = waitS \rightarrow \\
\quad\quad \mathsf{ti}(resp, t+1) = \langle\rangle \wedge stateS' = waitS \\
\quad\quad \wedge \\
\quad\quad (\mathsf{ti}(xchd, t) = \langle\rangle \rightarrow \mathsf{ti}(dataS, t+1) = \langle\rangle) \\
\quad\quad \wedge \\
\quad\quad (\mathsf{ti}(xchd, t) \neq \langle\rangle \rightarrow \mathsf{ti}(dataS, t+1) = \langle Decr(genKey^{-1}, xchd^t_{\mathsf{ft}})\rangle)
\end{array}
$$

where
$e0, e1, e2$ so that
$e0 = ungValue(init^t_{\mathsf{ft}}),$
$e1 = Enc(key(init^t_{\mathsf{ft}}), Sign(SKey^{-1}, \langle genKey, ungValue(init^t_{\mathsf{ft}}), key(init^t_{\mathsf{ft}})\rangle)),$
$e2 = Sign(CAKey^{-1}, \langle S, SKey\rangle)$

7 Related Work

See [11] for an overview on software engineering techniques for computer security. Another approach for formal development of secure systems is [15] which utilizes threat scenarios that are the result of threat identification and risk analysis and model those attacks that are of importance to the system's security. Other examples include the work reported in [22] (and the references there), which develops an approach for secure software engineering using the CASE tool AutoFocus. Other approaches for model-based development of security-critical systems include [4, 2, 21]; for a more detailed overview cf. [14]. Correct composition of specifications or models is generally considered a challenging task; cf. [16, 17, 5] for overviews and examples. Here we focus specifically on the (manual) composition of security-critical components and the question to what extent this preserves security properties. A related investigation was reported in [10]. Differences are that there, beharioural specifications were merged, while here we consider composition of component specifications (which however remain separate within the system specification, i.e. are not merged on a detailed level). Also, we specifically focus on the composition of security properties.

8 Conclusions

We present a methodology to represent cryptographic protocols and their composition properties in a formal way using the specification framework Focus. Having such a formal representation, one can argue about the protocol properties as well as the composition properties of different cryptographic protocols in a methodological way.

As a running example, a variant of the Internet security protocol TLS is presented. We analyzed the version of the protocol published in [3] and demonstrated a security flaw in this version using our approach. We also used the approach to harden the protocol in a formal way, and showed how to construct a secure channel on the basis of the corrected formal specification of the protocol. To achieve that, we also proved some general results regarding the composition of security properties in distributed systems.

On the base of such a specification one can then verify the protocol properties and their composition using the theorem prover Isabelle/HOL, as well as make automatic correctness proofs of syntactic interfaces for specified system components. Alternatively, we can translate it to a representation in the related CASE tool AutoFocus [20, 13] and use the simulation and model-checking facilities of this tool. This is the target of on-going work.

References

[1] *Refinement-based verification of interactive real-time systems.* ENTCS, 2008.

[2] M. Alam, M. Hafner, and R. Breu. Model-driven security engineering for trust management in SECTET. *Journal of Software*, 2(1), February 2007.

[3] V. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How much does it really cost? In *In Conference on Computer Communications (IEEE Infocom)*, pages 717–725. IEEE Computer Society, 1999.

[4] D.A. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, 2006.

[5] Jean Bézivin, Salim Bouzitouna, Marcos Didonet Del Fabro, Marie-Pierre Gervais, Frédéric Jouault, Dimitrios S. Kolovos, Ivan Kurtev, and Richard F. Paige. A canonical scheme for model composition. In Arend Rensink and Jos Warmer, editors, *ECMDA-FA*, volume 4066 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 2006.

[6] M. Broy. Compositional refinement of interactive systems. *J. ACM*, 44(6):850–891, 1997.

[7] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* Springer, 2001.

[8] Manfred Broy. Compositional refinement of interactive systems modelled by relations. *COMPOS'97: Revised Lectures from the International Symposium on Compositionality: The Significant Difference*, pages 130–149, 1998.

[9] Manfred Broy. Service-oriented systems engineering: Specification and design of services and layered architectures. The JANUS Approach. pages 47–81, July 2005.

[10] Greg Brunet, Marsha Chechik, and Sebastián Uchitel. Properties of behavioural model merging. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM*, volume 4085 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2006.

[11] P. Devanbu and S. Stubblebine. Software engineering for security: A roadmap. In *In 22nd International Conference on on Software Engineering (ICSE 2000): Future of Software Engineering Track*, pages 227–239. ACM, 2000.

[12] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983.

[13] Franz Huber, Bernhard Schätz, Alexander Schmidt, and Katharina Spies. AutoFocus - A Tool for Distributed Systems Specification. In *Proceedings FTRTFT'96 – Formal Techniques in Real-Time and Fault-Tolerant Systems*, number LNCS 1135, pages 467–470. Springer Verlag, 1996.

[14] J. Jürjens. *Secure Systems Development with UML.* Springer-Verlag, 2004.

[15] V. Lotz. Threat scenarios as a means to formally develop secure systems. *Journal of Computer Security*, 5(1):31–68, 1997.

[16] T. S. E. Maibaum. Mathematical foundations of software engineering: a roadmap. In *ICSE - Future of SE Track*, pages 161–172, 2000.

[17] Ana Milanova. Precise identification of composition relationships for uml class diagrams. In David F. Redmiles, Thomas Ellman, and Andrea Zisman, editors, *ASE*, pages 76–85. ACM, 2005.

[18] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[19] M. Spichkova. *Specification and Seamless Verification of Embedded Real-Time Systems: FOCUS on Isabelle*. PhD thesis, Technische Universität München, 2007.

[20] Technische Universität München. AutoFocus 2, 2007. `http://www4.informatik.tu-muenchen.de/~af2`.

[21] Jon Whittle, Duminda Wijesekera, and Mark Hartong. Executable misuse cases for modeling security concerns. In Robby, editor, *ICSE*, pages 121–130. ACM, 2008.

[22] G. Wimmel. *Model-based Development of Security-Critical Systems*. PhD thesis, Technische Universität München, 2005.