TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

Component Composition: Formal Specification and Verification of Cryptographic Properties

Maria Spichkova

TUM-I124

Component Composition: Formal Specification and Verification of Cryptographic Properties

Maria Spichkova

Institut für Informatik, Technische Universität München

Boltzmannstr. 3, 85748 Garching, Germany

Abstract

This paper presents an optimized and refined methodology to specify crypto-based distributed software and to verify their composition properties in a formal way.

We suggest to specify all components in Focus, a framework for formal specification and development of interactive systems. Having a formal Focus representation of a protocol components, one can argue about their properties and composition in a methodological way, referring to the approach "Focus on Isabelle" and checking the defined properties formal proofs using the theorem prover Isabelle/HOL, as well as make automatic correctness proofs of syntactic interfaces for specified system components.

As a running example, a variant of the Internet security protocol TLS is presented. We analyzed one of the versions of the protocol using refined Focus specification and demonstrated a security flaw in this version formally, using Isabelle/HOL. We also used the extended approach to harden the protocol in a formal way, and showed how to construct a new version of the secure channel on the basis of the corrected formal specification of the protocol. The formal proof that the discussed flaw no more exist in this corrected version of the protocol was done also in Isabelle/HOL.

On the base of these protocol we specified secure channels that adopt the main protocol properties.

Keywords: Formal Specification, Verification, Composition, Cryptographic Properties

Contents

1 Introduction

In this paper we discuss a result of extension and optimization of the draft ideas presented in [SJ08]: the question how we can combine system components, which enforce a particular security requirement in a way that allows us to predict which properties the combined system will have, is very important and very difficult to answer. Thus, we need a methodology that allows us not only to represent crypto-based software and their composition properties in a formal way, but also to argue about them (semi)automatically, using theorem provers – the paper-and-pencil proofs are not enough for this case. Therefore, an extension to the representation which is suitable to the theorem prover is essential.

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, and applying the methodology "Focus on Isabelle" [Spi07] we can verify the properties and their combination using the Isabelle/HOL theorem prover [NPW02]. Using "Focus on Isabelle" 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 it we can perform automatic correctness proofs of syntactic interfaces for specified system components.

Using the extended approach, we can, as before, demonstrate a security flaw in the protocol and show how to prove security properties of a corrected version, but now we can do it not only in paper-and-pencil version but also using more strict and solid way: a semiautomatic theorem prover. We also transfer to the extended version of the approach the idea of secure channels and provide some general results on composition of security properties.

Like in [SJ08], we use here as a running example a variant of the Internet security protocol TLS published in [APS99], but the Focus specification of the protocol is now corrected and refined to be more readable and to avoid misinterpretation. Thus, in contrast to [SJ08] we used here an optimized method to specify components in Focus, some of the optimization ideas were previously discussed in [Spi11a] and [Spi11b].

2 Focus: Composition of Components

Focus [BS01] 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. 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 $(I_P \rhd O_P)$ is used. The formal meaning of a specification is exactly this external *input/output relation*.

The central concept of this framework 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. 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).

A 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*. 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.

A large number of composition properties defined in [BS01] can be represented in Isabelle/HOL according the rules we introduce in [Spi07] to prove them automatically. By representing protocols as Focus specifications, like discussed in [SJ08], we can describe them as components or services (see [BS01, Bro05]) and can argue about properties of component compositions using well-founded theories of component- and service-composition (see [Bro97, Bro98]). Thus, using this representation we can combine different components involved in a protocol and can check in Isabelle/HOL whether this combination satisfies the desired security properties. In such a way 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"[Spi07].

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.

See [BS01] and [Spi07] for more background on Focus and its extensions.

As mentioned in [SJ08], by representing protocols as FOCUS specifications we can describe them as components or services (see [BS01, Bro05]) and can argue about properties of component compositions using well-founded theories of component- and service-composition (see [Bro97, Bro98]).

The FOCUS semantics of a *composite* specification $S = S_1 \otimes \cdots \otimes S_n$ is defined in [BS01] as follows:

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

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.

A large number of composition properties defined in [BS01, Spi07] can be represented in Isabelle/HOL according the rules we introduce in [Spi07] to prove them automatically. Thus, using this representation we can combine different components involved in a protocol and can check in Isabelle/HOL whether this combination satisfies the desired security properties . 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"[Spi07].

3 Secrecy: Focus on Isabelle

In this section we introduce an Isabelle/HOL formalization of the security property of data secrecy, the corresponding definitions, and a number of abstract data types used in this formalization. This formalization is a translation of the FOCUS representation of these artifacts (see [SJ08]).

3.1 Data Types

We assume here 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:

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

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 [DY83]):

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

The corresponding definition in Isabelle:

consts

Enc :: "*Keys* ⇒ *Expression list* ⇒ *Expression list*"

Decr :: "*Keys* ⇒ *Expression list* ⇒ *Expression list*"

Sign :: "*Keys* ⇒ *Expression list* ⇒ *Expression list*"

Ext :: "*Keys* ⇒ *Expression list* ⇒ *Expression list*"

EncrDecrKeys :: "*Keys* ⇒ *Keys* ⇒ *bool*"

axioms

ExtSign :

"*EncrDecrKeys K*1 *K*2 → (*Ext K*1 (*Sign K*2 *E*)) = *E*" *DecrEnc* :

"*EncrDecrKeys K*1 *K*2 → (*Decr K*2 (*Enc K*1 *E*)) = *E*"

We denote by $K_P \subseteq Keys$ and $S_P \subseteq Secret$ 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.

In Isabelle we define this as follows:

consts

$specKeys :: \text{“}specID \Rightarrow Keys\ set\text{”}$

consts

$specSecrets :: \text{“}specID \Rightarrow Secrets\ set\text{”}$

constdefs

$specKeysSecrets :: \text{“}specID \Rightarrow KS\ set\text{”}$

$\text{“}specKeysSecrets\ C \equiv$

$\{y.\, \exists\, x.y = (kKS\ x) \wedge (x \in (specKeys\ C))\} \cup$

$\{z.\, \exists\, s.z = (sKS\ s) \wedge (s \in (specSecrets\ C))\}\text{”}$

3.2 Correct Composition

We assume in our specification that the composition of components has a number of general properties which sometimes seem to be obvious, but for a formal representation is essential to mention these properties explicitly either we can't make the proofs in a correct way.

The sets of private keys and unguessable values used by a composed component $C = C_1 \otimes \cdots \otimes C_n$ must be defined by union of corresponding sets. In Isabelle/HOL we define this by the following predicates (according the general ideas presented in [Spi07]):

constdefs

$correctCompositionKeys :: \text{“}specID \Rightarrow bool\text{”}$

$\text{“}correctCompositionKeys\ x \equiv$

$subcomponents\ x \neq \{\} \rightarrow specKeys\ x = \bigcup\, (specKeys\ \text{`}\ (subcomponents\ x))\text{”}$

constdefs

$correctCompositionSecrets :: \text{“}specID \Rightarrow bool\text{”}$

$\text{“}correctCompositionSecrets\ x \equiv$

$subcomponents\ x \neq \{\} \rightarrow specSecrets\ x = \bigcup\, (specSecrets\ \text{`}\ (subcomponents\ x))\text{”}$

constdefs

$correctCompositionKS :: \text{“}specID \Rightarrow bool\text{”}$

$\text{“}correctCompositionKSx \equiv$

$subcomponents\ x \neq \{\} \rightarrow$

$specKeysSecrets\ x = \bigcup\, (specKeysSecrets\ \text{`}\ (subcomponents\ x))\text{”}$

The following properties must hold for the correct composed components:

- If xb is a private key of the composed component C, then this key must belong to the set of private keys of one subcomponents of C.

  $$C = C_1 \otimes \cdots \otimes C_n \wedge xb \in K_C \rightarrow \exists\, i \in [1..n].\ xb \in K_{C_i}$$

  In Isabelle we can represent this property by the following lemma

  $$\text{“}[\![\mathit{correctCompositionKeys}\ C;\ x \in \mathit{subcomponents}\ C;\ xb \in \mathit{specKeys}\ C]\!] \Rightarrow \exists\, x \in \mathit{subcomponents}\ C.\ xb \in \mathit{specKeys}\ x\text{”}$$

  or more general:

  $$\text{“}[\![\mathit{correctCompositionKS}\ C;\ x \in \mathit{subcomponents}\ C;\ xa \in \mathit{specKeys}\ C]\!] \Rightarrow \exists\, x \in \mathit{subcomponents}\ C.\ xa \in \mathit{specKeys}\ x\text{”}$$

- If xb is an unguessable value used by the composed component C, then this value must belong to the set of unguessable values used by one subcomponents of C.

  $$C = C_1 \otimes \cdots \otimes C_n \wedge xb \in S_C \rightarrow \exists\, i \in [1..n].\ xb \in S_{C_i}$$

  In Isabelle we can represent this property by the following lemma

  $$\text{“}[\![\mathit{correctCompositionSecrets}\ C;\ x \in \mathit{subcomponents}\ C;\ s \in \mathit{specSecrets}\ C]\!] \Rightarrow \exists\, x \in \mathit{subcomponentsC}.\ s \in \mathit{specSecrets}\ x\text{”}$$

  or more general:

  $$\text{“}[\![\mathit{correctCompositionKS}\ C;\ x \in \mathit{subcomponents}\ C;\ xa \in \mathit{specSecrets}\ C]\!] \Rightarrow \exists\, x \in \mathit{subcomponents}\ C.\ xa \in \mathit{specSecrets}\ x\text{”}$$

- If xb is a private key of one subcomponents of the composed component C, then this key must belong to the set of private keys of C.

  $$C = C_1 \otimes \cdots \otimes C_n \wedge 1 \leq i \leq n \wedge xb \in K_{C_i} \rightarrow xb \in K_C$$

  $$\text{“}[\![\mathit{correctCompositionKeys}\ C;\ x \in \mathit{subcomponents}\ C;\ xc \in \mathit{specKeys}\ x]\!] \Rightarrow xc \in \mathit{specKeysC}\text{”}$$

  $$\text{“}[\![\mathit{correctCompositionKS}\ C;\ x \in \mathit{subcomponents}\ C;\ xa \in \mathit{specKeys}\ x]\!] \Rightarrow xa \in \mathit{specKeys}\ C\text{”}$$

- If xb is an unguessable value used by one subcomponents of the composed component C, then this value must belong to the set of unguessable values used by C.

  $C = C_1 \otimes \cdots \otimes C_n \wedge 1 \leq i \leq n \wedge xb \in S_{C_i} \rightarrow xb \in S_C$

  "$[\![correctCompositionSecrets\ C;\ x \in subcomponents\ C;\ xc \in specSecrets\ x]\!]$
  $\Rightarrow xc \in specSecrets\ C$"

  "$correctCompositionKeys\ C\ \wedge correctCompositionSecrets\ C$
  $= correctCompositionKS\ C$"

- If xb does not belong to the set of private keys and unguessable values of any subcomponent of the composed component $PQ = P \otimes Q$, then xp does not belong to the set of private keys and unguessable values of PQ.

  $PQ = P \otimes Q \wedge xb \notin KS_P \wedge xb \notin KS_Q \rightarrow xb \notin KS_{PQ}$

  "$[\![subcomponents\ PQ = \{P, Q\};\ correctCompositionKS\ PQ;$
  $ks \notin specKeysSecrets\ P;\ ks \notin specKeysSecrets\ Q]\!]$
  $\Rightarrow ks \notin specKeysSecrets\ PQ$"

We also add to the set of properties of composition the following two lemmas:

- If a channel x belongs to the set of input channels of the composition $PQ = P \otimes Q$ for any two components P and Q, then this channel must belong to the set of input channels of P or Q.

  $x \in i_{P \otimes Q} \rightarrow x \in i_P \vee x \in i_Q$

  "$[\![subcomponents\ PQ = \{P, Q\};\ correctCompositionIn\ PQ;\ x \in ins\ PQ]\!]$
  $\Rightarrow x \in ins\ P \vee x \in ins\ Q$"

- If a channel x belongs to the set of output channels of the composition $PQ = P \otimes Q$ for any two components P and Q, then this channel must belong to the set of output channels of P or Q.

  $x \in o_{P \otimes Q} \rightarrow x \in o_P \vee x \in o_Q$

  "$[\![subcomponents\ PQ = \{P, Q\};\ correctCompositionOut\ PQ;\ x \in out\ PQ]\!]$
  $\Rightarrow x \in out\ P \vee x \in out\ Q$"

3.3 New Auxiliary Predicates

To discuss the next propositions we introduce first of all a number of new predicates.

A channel $ch \in i_P$ of a component P is a single input channel of this component that may eventually input an expression $E \in CExp$ (denoted by $exprChannelSingleI(P, ch, E)$ in Focus and by *ine_exprChannelSingle sP ch E* in Isabelle/HOL):

$$exprChannelSingleI(P, ch, E) \stackrel{\mathsf{def}}{=}$$
$$ch \in i_P \wedge (\exists\, t \in \mathbb{N} : \; E \in \mathsf{ti}(ch, t)) \wedge$$
$$\forall\, x \in i_P : \; x \neq ch \rightarrow \forall\, t \in \mathbb{N} : \; E \notin \mathsf{ti}(x, t)$$

The corresponding definition in Isabelle:

constdefs

ine_exprChannelSingle :: "*specID* ⇒ *chanID* ⇒ *Expression* ⇒ *bool*"

"*ine_exprChannelSinglesPchE* ≡ (*ch* ∈ (*ins sP*)) ∧ (*exprChannel ch E*) ∧

∀(*x* :: *chanID*)(*t* :: *nat*).(*x* ∈ *ins sP* ∧ *x* ≠ *ch* → ¬(*exprChannel x E*))"

The Focus predicate $exprChannelSetI(P, chSet, E)$ yields true if only the channels from the set $chSet$, which is a subset of input channels of a component P, may eventually input an expression $E \in CExp$:

$$exprChannelSetI(P, chSet, E) \stackrel{\mathsf{def}}{=}$$
$$\forall\, x : \; x \in chSet \rightarrow ch \in i_P \wedge (\exists\, t \in \mathbb{N} : \; E \in \mathsf{ti}(ch, t)) \wedge$$
$$\forall\, x : \; x \notin chSet \wedge ch \in i_P \rightarrow (\forall\, t \in \mathbb{N} : \; E \notin \mathsf{ti}(x, t))$$

The corresponding definition in Isabelle:

constdefs

ine_exprChannelSet :: "*specID* ⇒ *chanIDset* ⇒ *Expression* ⇒ *bool*"

"*ine_exprChannelSet sP chSet E* ≡

((∀(*x* :: *chanID*).(*x* ∈ *chSet* → (*x* ∈ *ins sP* ∧ *exprChannel x E*))) ∧

(∀(*x* :: *chanID*).(*x* ∉ *chSet* ∧ *x* ∈ *ins sP* → ¬(*exprChannel x E*))))"

A channel $ch \in o_P$ of a component P is a single output channel of this component that may eventually output an expression $E \in CExp$ (denoted by $exprChannelSingleO(P, ch, E)$ in Focus and by *out_exprChannelSingle sP ch E* in Isabelle/HOL):

$$exprChannelSingleO(P, ch, E) \stackrel{\mathsf{def}}{=}$$
$$ch \in o_P \wedge (\exists\, t \in \mathbb{N} : \; E \in \mathsf{ti}(ch, t)) \wedge$$
$$\forall\, x \in o_P : \; x \neq ch \rightarrow \forall\, t \in \mathbb{N} : \; E \notin \mathsf{ti}(x, t)$$

The corresponding definition in Isabelle:

constdefs

$out\_exprChannelSingle$ :: "$specID \Rightarrow chanID \Rightarrow Expression \Rightarrow bool$"
"$out\_exprChannelSingle\ sP\ ch\ E \equiv$
$(ch \in out\ sP) \wedge (exprChannel\ ch\ E) \wedge$
$\forall (x :: chanID)(t :: nat).(x \in outsP \wedge x \neq ch \rightarrow \neg(exprChannel\ x\ E))$"

The FOCUS predicate $exprChannelSetO(P, chSet, E)$ yields true if only the channels from the set $chSet$, which is a subset of output channels of a component P, may eventually output an expression $E \in CExp$:

$$exprChannelSetO(P, chSet, E) \stackrel{\mathsf{def}}{=}$$
$$\forall x : \ x \in chSet \rightarrow ch \in o_P \wedge (\exists t \in \mathbb{N} : \ E \in \mathsf{ti}(ch, t)) \wedge$$
$$\forall x : \ x \notin chSet \wedge ch \in o_P \rightarrow (\forall t \in \mathbb{N} : \ E \notin \mathsf{ti}(x, t))$$

The corresponding definition in Isabelle:

constdefs

$out\_exprChannelSet$ :: "$specID \Rightarrow chanIDset \Rightarrow Expression \Rightarrow bool$"
"$out\_exprChannelSet\ sP\ chSet\ E \equiv$
$((\forall (x :: chanID).(x \in chSet \rightarrow (x \in out\ sP \wedge (exprChannelxE)))) \wedge$
$(\forall (x :: chanID).(x \notin chSet \wedge x \in out\ sP \rightarrow \neg(exprChannelxE))))$"

Now we present a number of properties that show the relation between these predicates:

$$exprChannelSingleI(P, ch, E) \rightarrow exprChannelSetI(P, \{ch\}, E)$$
$$exprChannelSingleO(P, ch, E) \rightarrow exprChannelSetO(P, \{ch\}, E)$$
$$exprChannelSetI(P, \{ch\}, E) \rightarrow exprChannelSingleI(P, ch, E)$$
$$exprChannelSetO(P, \{ch\}, E) \rightarrow exprChannelSingleO(P, ch, E)$$

"$[\![ine\_exprChannelSingle\ P\ ch\ E]\!] \Rightarrow ine\_exprChannelSet\ P\ \{ch\}\ E$"
"$[\![out\_exprChannelSingle\ P\ ch\ E]\!] \Rightarrow out\_exprChannelSet\ P\ \{ch\}\ E$"
"$[\![ine\_exprChannelSet\ P\ \{ch\}\ E]\!] \Rightarrow ine\_exprChannelSingle\ P\ ch\ E$"
"$[\![out\_exprChannelSet\ P\ \{ch\}\ E]\!] \Rightarrow out\_exprChannelSingle\ P\ ch\ E$"

3.4 Input and Output of Expressions

In this Section we refine and optimize the definition presented in [SJ08] and represent them in Isabelle/HOL. We omit now proofs for the all discussed here proposition and theorems: paper-and-pensil proofs of them are shown in [SJ08], the semi-automatic proofs are given in the corresponding Isabelle/HOL theories we created (Secrecy.thy, Secrecy_types.thy).

Please note that we use for this purpose the ideas from [Spi07], where the argumentation and proofs about the syntactical interface are represented separately from the main part of the specification to allow automatic verification of syntax correctness. Like in [Spi07] we use here the following notation: sC denotes a (syntactical) identifier of a component C, ch_x denotes a (syntactical) identifier of a channel x. The predicates *correctCompositionIn*, *correctCompositionOut*, *correctCompositionLoc* etc. are defined in Isabelle/HOL to insure that a composition of components has the same properties as discussed in [BS01, Spi07], i.e. that the composition is done in a correct way.

Let assume a component P without any sheaves of channels, $(I_P \rhd O_P)$ with $i_P = \{x_1, \dots, x_n\}$ and $o_P = \{y_1, \dots, y_m\}$.

Corresponding notations in Isabelle/HOL are the following ones: $ins\ sP = \{ch\_x_1, \dots, ch\_x_n\}$ and $out\ sP = \{ch\_y_1, \dots, ch\_y_m\}$.

The set $l_P = \{l_1, \dots, l_z\}$ of local channels of component P is represented in Isabelle/HOL as follows: $loc\ sP = \{ch\_l_1, \dots, ch\_l_z\}$.

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

$$P^{\mathsf{eout}}(E) \stackrel{\mathsf{def}}{=} P(x_1, \dots, x_n, y_1, \dots, y_m) \;\wedge\; \exists\, s \in o_P : \; \exists\, t \in \mathbb{N} : \; E \in \mathsf{ti}(s, t)$$

The corresponding representation in Isabelle:

consts

exprChannel :: "*chanID* ⇒ *Expression* ⇒ *bool*"

constdefs

eout :: "*specID* ⇒ *Expression* ⇒ *bool*"

"*eout sP E* ≡

∃ *ch* :: *chanID*. (*ch* ∈ (*out sP*)) ∧ (*exprChannel ch E*)"

A component P, $(I_P \rhd O_P)$, may eventually output an expression $E \in CExp$ via M (denoted by $P_M^{\mathsf{eout}}(E)$ in Focus and by $eoutM\ sP\ M\ E$ in Isabelle/HOL) 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:

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

The corresponding representation in Isabelle:

constdefs
eoutM :: "*specID* ⇒ *chanID set* ⇒ *Expression* ⇒ *bool*"
"*eoutM sP M E* ≡
∃ *ch* :: *chanID*. ((*ch* ∈ (*out sP*)) ∧ (*ch* ∈ *M*) ∧ (*exprChannel ch E*))

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

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

The corresponding representation in Isabelle:

constdefs
ine :: "*specID* ⇒ *Expression* ⇒ *bool*"
"*ine sP E* ≡
∃ *ch* :: *chanID*. ((*ch* ∈ (*ins sP*)) ∧ (*exprChannel ch E*))"

A component P, $(I_P \rhd O_P)$, may eventually get an expression $E \in \mathit{CExp}$ via M (denoted by $P^{\mathsf{ine}}_M(E)$ in Focus and by *ineM sP M E* in Isabelle/HOL) 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:

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

The corresponding representation in Isabelle:

constdefs
ineM :: "*specID* ⇒ *chanID set* ⇒ *Expression* ⇒ *bool*"
"*ineM sP M E* ≡
∃ *ch* :: *chanID*. ((*ch* ∈ (*ins sP*)) ∧ (*ch* ∈ *M*) ∧ (*exprChannel ch E*))"

Please note that the following properties hold for these predicates:

$$\mathit{exprChannelSetI}(P, \mathit{ChSet}, E) \wedge \mathit{ChSet} \neq \{\} \rightarrow P^{\mathsf{ine}}(E)$$
$$\mathit{exprChannelSetI}(P, \mathit{ChSet}, E) \wedge \mathit{ChSet} = \{\} \rightarrow \neg P^{\mathsf{ine}}(E)$$

The corresponding representation of these properties in Isabelle by lemmas:

"⟦*ine_exprChannelSet P ChSet E*; *ChSet* ≠ {}⟧ ⇒ *ine P E*"
"⟦*ine_exprChannelSet P ChSet E*; *ChSet* = {}⟧ ⇒ ¬(*ine P E*)"

We omit here the discussion of a number of other auxiliary lemmas we defined on these predicates, because they do not belong directly to the properties of component composition, and continue with the presentation of the input and output properties of the composed components.

3.5 Composing Input Properties

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* $m \notin KS_Q$*):*

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

The corresponding representation in Isabelle:

$$\text{“}[\![\ ine\ PQ\ E;\ subcomponents\ PQ\ =\ P, Q;\ correctCompositionIn\ PQ\ ]\!]$$
$$\Rightarrow\ ine\ P\ E\ \vee\ ine\ Q\ E\text{”}$$

$$\text{“}[\![\ ineM\ PQ\ M\ E;\ subcomponents\ PQ\ =\ P, Q;\ correctCompositionIn\ PQ\ ]\!]$$
$$\Rightarrow\ ineM\ P\ M\ E\ \vee\ ineM\ Q\ M\ E\text{”}$$

□

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* $m \notin KS_Q$*):*

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

The corresponding representation in Isabelle:

$$\text{“}[\![\ eout\ PQ\ E;\ subcomponents\ PQ\ =\ \{P, Q\};\ correctCompositionOut\ PQ\ ]\!]$$
$$\Rightarrow\ eout\ P\ E\ \vee\ eout\ Q\ E\text{”}$$

$$\text{“}[\![\ eoutM\ PQ\ M\ E;\ subcomponents\ PQ\ =\ \{P, Q\};\ correctCompositionOut\ PQ\ ]\!]$$
$$\Rightarrow\ eoutM\ P\ M\ E\ \vee\ eoutM\ Q\ M\ E\text{”}$$

□

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* $m \notin KS_Q$*):*

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

The corresponding representation in Isabelle:

$$\text{“}[\![\ \neg\ (ine\ P\ E);\ \neg\ (ine\ Q\ E);\ subcomponents\ PQ\ =\ P, Q;$$
$$correctCompositionIn\ PQ\ ]\!]$$
$$\Rightarrow\ \neg\ (ine\ PQ\ E)\text{”}$$

"$\llbracket\ \neg\,(ineM\ \ P\ \ M\ \ E);\ \ \neg\,(ineM\ \ Q\ \ M\ \ E);\ \ subcomponents\ \ PQ\ \ =\ \ P, Q;$
$correctCompositionIn\ \ PQ\ \rrbracket$
$\Rightarrow\ \neg\,(ineM\ \ PQ\ \ M\ \ E)$"

□

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:*

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

□

In addition to the paper-and-pensil proof from [SJ08] we can easily find a counterexample in Isabelle to show that the properties above do not hold in general, but even more important is to find out for which special cases these properties hold and for which ones hold exactly opposite properties. Thus, we extend the set of proven properties by a number of extra propositions.

Proposition 1 *For any components P and Q the following property of the parallel composition $P \otimes Q$, i.e. with an empty set of local channels, holds ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$):*

$$(P^{ine}(e)\ \vee\ Q^{ine}(e)) \wedge l_{P \otimes Q} = \{\}\ \rightarrow\ (P \otimes Q)^{ine}(e)$$

The corresponding representation in Isabelle:

"$\llbracket\ (ine\ \ P\ \ E) \vee (ine\ \ Q\ \ E);\ subcomponents\ \ PQ = \{P, Q\};$
$correctCompositionIn\ \ PQ;\ loc\ \ PQ = \{\}\ \rrbracket$
$\Rightarrow\ ine\ \ PQ\ \ E$"

□

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

$$P^{\mathit{ine}}(e) \wedge \exists\, ch.(ch \in i_P \wedge ch \notin l_{P\otimes Q} \wedge (\exists\, t \in \mathbb{N}: \; e \in \mathit{ti}(ch,t)))$$
$$\to \; (P \otimes Q)^{\mathit{ine}}(e)$$

$$(P^{\mathit{ine}}(e) \vee Q^{\mathit{ine}}(e)) \wedge$$
$$\exists\, ch.((ch \in i_P \vee ch \in i_Q) \wedge ch \notin l_{P\otimes Q} \wedge (\exists\, t \in \mathbb{N}: \; e \in \mathit{ti}(ch,t)))$$
$$\to \; (P \otimes Q)^{\mathit{ine}}(e)$$

$$P^{\mathit{ine}}_M(e) \wedge \exists\, ch.(ch \in i_P \wedge ch \in M \wedge ch \notin l_{P\otimes Q} \wedge (\exists\, t \in \mathbb{N}: \; e \in \mathit{ti}(ch,t)))$$
$$\to \; (P \otimes Q)^{\mathit{ine}}_M(e)$$

$$(P^{\mathit{ine}}_M(e) \vee Q^{\mathit{ine}}_M(e)) \wedge$$
$$\exists\, ch.((ch \in i_P \vee ch \in i_Q) \wedge ch \in M \wedge ch \notin l_{P\otimes Q} \wedge (\exists\, t \in \mathbb{N}: \; e \in \mathit{ti}(ch,t)))$$
$$\to \; (P \otimes Q)^{\mathit{ine}}_M(e)$$

The corresponding representation in Isabelle:

$$\text{“}[\![\; \mathit{ine}\ P\ E;\ \mathit{subcomponents}\ PQ = \{P, Q\};\ \mathit{correctCompositionIn}\ PQ;$$
$$\exists\, ch.((ch \in \mathit{ins}\ P) \;\wedge\; (ch \notin \mathit{loc}\ PQ) \;\wedge\; \mathit{exprChannel}\ ch\ E) \;]\!]$$
$$\Rightarrow\; \mathit{ine}\ PQ\ E\text{”}$$

$$\text{“}[\![\; \mathit{ineM}\ P\ M\ E;\ \mathit{subcomponents}\ PQ = \{P, Q\};\ \mathit{correctCompositionIn}\ PQ;$$
$$\exists\, ch.((ch \in \mathit{ins}\ Q) \wedge ch \in M \wedge ch \notin \mathit{loc}\ PQ \wedge \mathit{exprChannel}\ ch\ E)]\!]$$
$$\Rightarrow\; \mathit{ineM}\ PQ\ M\ E\text{”}$$

$$\text{“}[\![\; ((\mathit{ine}\ P\ E) \vee (\mathit{ine}\ Q\ E));\ \mathit{subcomponents}\ PQ = \{P, Q\};$$
$$\mathit{correctCompositionIn}\ PQ;$$
$$\exists\, ch.((ch \in \mathit{ins}\ P \vee ch \in \mathit{ins}\ Q) \;\wedge\; (\mathit{exprChannel}\ ch\ E) \;\wedge\; (ch \notin (\mathit{loc}\ PQ)))]\!]$$
$$\Rightarrow\; \mathit{ine}\ PQ\ E\text{”}$$

$$\text{“}[\![\; (\mathit{ineM}\ P\ M\ E) \vee (\mathit{ineM}\ Q\ M\ E);\ \mathit{subcomponents}\ PQ = \{P, Q\};$$
$$\mathit{correctCompositionIn}\ PQ;$$
$$\exists\, ch.((ch \in \mathit{ins}\ P \vee ch \in \mathit{ins}\ Q) \wedge ch \in M \wedge \mathit{exprChannel}\ ch\ E \wedge ch \notin \mathit{loc}\ PQ)]\!]$$
$$\Rightarrow\; \mathit{ineM}\ PQ\ M\ E\text{”}$$

□

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

$$P^{ine}(E) \land \neg Q^{ine}(E) \land \exists\, ch \in l_{P \otimes Q} : exprChannelSingleI(P, ch, E) \\ \to\ (P \otimes Q)^{ine}(E)$$

$$P^{ine}_M(E) \land \neg Q^{ine}_M(E) \land \exists\, ch \in l_{P \otimes Q} : (exprChannelSetI(P, ch, E) \land ch \in M) \\ \to\ (P \otimes Q)^{ine}_M(E)$$

The corresponding representation in Isabelle:

"$[\![$ $ine\ P\ E$; $\neg ine\ Q\ E$; $subcomponents\ PQ = \{P, Q\}$; $correctCompositionIn\ PQ$;
$\exists\ ch.((ine\_exprChannelSingle\ P\ ch\ E) \land (ch \in loc\ PQ))]\!]$
$\Rightarrow\ \neg\ (ine\ PQ\ E)$"

"$[\![$ $ineM\ P\ M\ E$; $\neg\ (ineM\ Q\ M\ E)$; $subcomponents\ PQ = \{P, Q\}$;
$correctCompositionIn\ PQ$;
$\exists\ ch.((ine\_exprChannelSingle\ P\ ch\ E) \land (ch \in M) \land (ch \in loc\ PQ))]\!]$
$\Rightarrow\ \neg\ (ineM\ PQ\ M\ E)$"

□

Proposition 4 *For any components P and Q the following properties of the composition $P \otimes Q$ ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$) hold:*

$$\neg Q^{ine}(E) \land exprChannelSetI(P, ChSet, E) \land \forall\, ch : ch \in ChSet \to ch \in l_{P \otimes Q} \\ \to\ \neg(P \otimes Q)^{ine}(E)$$

$$\neg Q^{ine}_M(E) \land exprChannelSetI(P, ChSet, E) \land \forall\, ch : ch \in ChSet \to ch \in l_{P \otimes Q} \\ \to\ \neg(P \otimes Q)^{ine}_M(E)$$

The corresponding representation in Isabelle:

"$[\![$ $\neg(ine\ Q\ E)$; $subcomponents\ PQ = \{P, Q\}$; $correctCompositionIn\ PQ$;
$ine\_exprChannelSet\ P\ ChSet\ E$;
$\forall(x :: chanID).((x \in ChSet) \to (x \in loc\ PQ))]\!]$
$\Rightarrow\ \neg(ine\ PQ\ E)$"

"$[\![$ $\neg(ineM\ Q\ M\ E)$; $subcomponents\ PQ = \{P, Q\}$;
$correctCompositionIn\ PQ$; $ine\_exprChannelSet\ P\ ChSet\ E$;
$\forall(x :: chanID).((x \in ChSet) \to (x \in loc\ PQ))]\!]$
$\Rightarrow \neg(ineM\ PQ\ M\ E)$"

□

Proposition 5 *For any components P and Q the following properties of the composition $P \otimes Q$ ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$) hold:*

$$exprChannelSetI(P, ChSetP, E) \wedge exprChannelSetI(Q, ChSetQ, E) \wedge$$
$$\forall\, ch : ch \in ChSetP \rightarrow ch \in l_{P \otimes Q} \;\; \wedge \;\; \forall\, ch : ch \in ChSetQ \rightarrow ch \in l_{P \otimes Q}$$
$$\rightarrow \; \neg (P \otimes Q)^{\mathsf{ine}}(E)$$

$$exprChannelSetI(P, ChSetP, E) \wedge exprChannelSetI(Q, ChSetQ, E) \wedge$$
$$M = ChSetP \cup ChSetQ \; \wedge$$
$$\forall\, ch : ch \in ChSetP \rightarrow ch \in l_{P \otimes Q} \;\; \wedge \;\; \forall\, ch : ch \in ChSetQ \rightarrow ch \in l_{P \otimes Q}$$
$$\rightarrow \; \neg (P \otimes Q)^{\mathsf{ine}}_M(E)$$

The corresponding representation in Isabelle:

"⟦ *subcomponents PQ* = {*P*, *Q*}; *correctCompositionIn PQ*;
ine_exprChannelSet P ChSetP E; *ine_exprChannelSet Q ChSetQ E*;
∀(*x* :: *chanID*).((*x* ∈ *ChSetP*) → (*x* ∈ *loc PQ*));
∀(*x* :: *chanID*).((*x* ∈ *ChSetQ*) → (*x* ∈ *loc PQ*))⟧
⟹ ¬(*ine PQ E*)"

"⟦ *subcomponents PQ* = {*P*, *Q*}; *correctCompositionIn PQ*;
ine_exprChannelSet P ChSetP E; *ine_exprChannelSet Q ChSetQ E*;
M = *ChSetP* ∪ *ChSetQ*;
∀(*x* :: *chanID*).((*x* ∈ *ChSetP*) → (*x* ∈ (*loc PQ*)));
∀(*x* :: *chanID*).((*x* ∈ *ChSetQ*) → (*x* ∈ (*loc PQ*)))⟧
⟹ ¬(*ineM PQ M E*)"

□

3.6 Composing Output Properties

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:*

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

□

In addition to the paper-and-pensil proof from [SJ08] we can easily find a counterexample in Isabelle to show that the properties above do not hold in general, but even more important is to find out for which special cases these properties hold and for which ones hold exactly opposite properties. Thus, we extend the set of proven properties by a number of extra propositions.

Proposition 6 *For any components P and Q the following property of the parallel composition $P \otimes Q$, i.e. with an empty set of local channels, holds ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$):*

$$(P^{\mathsf{eout}}(e) \ \vee \ Q^{\mathsf{eout}}(e)) \wedge l_{P \otimes Q} = \{\} \ \rightarrow \ (P \otimes Q)^{\mathsf{eout}}(e)$$

The corresponding representation in Isabelle:

$$\text{“}[\![(eout\ P\ E) \vee (eout\ Q\ E);$$
$$subcomponents\ PQ = \{P, Q\};\ correctCompositionOut\ PQ;\ loc\ PQ = \{\}]\!]$$
$$\Rightarrow\ eout\ PQ\ E\text{”}$$

□

Proposition 7 *For any components P and Q the following properties of the composition $P \otimes Q$ ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$) hold:*

$$P^{\mathsf{eout}}(e) \wedge \exists\, ch.(ch \in o_P \wedge ch \notin l_{P \otimes Q} \wedge (\exists\, t \in \mathbb{N} :\ e \in \mathsf{ti}(ch, t)))$$
$$\rightarrow\ (P \otimes Q)^{\mathsf{eout}}(e)$$

$$P_M^{\mathsf{eout}}(e) \wedge \exists\, ch.(ch \in o_P \wedge ch \in M \wedge ch \notin l_{P \otimes Q} \wedge (\exists\, t \in \mathbb{N} :\ e \in \mathsf{ti}(ch, t)))$$
$$\rightarrow\ (P \otimes Q)_M^{\mathsf{eout}}(e)$$

$$(P^{\mathsf{eout}}(e) \vee Q^{\mathsf{eout}}(e)) \wedge$$
$$\exists\, ch.((ch \in o_P \vee ch \in o_Q) \wedge ch \notin l_{P \otimes Q} \wedge (\exists\, t \in \mathbb{N} :\ e \in \mathsf{ti}(ch, t)))$$
$$\rightarrow\ (P \otimes Q)^{\mathsf{eout}}(e)$$

$$(P_M^{\mathsf{eout}}(e) \vee Q_M^{\mathsf{eout}}(e)) \wedge$$
$$\exists\, ch.((ch \in o_P \vee ch \in o_Q) \wedge ch \in M \wedge ch \notin l_{P \otimes Q} \wedge (\exists\, t \in \mathbb{N} :\ e \in \mathsf{ti}(ch, t)))$$
$$\rightarrow\ (P \otimes Q)_M^{\mathsf{eout}}(e)$$

The corresponding representation in Isabelle:

$$\text{“}[\![eout\ P\ E;$$
$$subcomponents\ PQ = \{P, Q\};\ correctCompositionOut\ PQ;$$
$$\exists\, ch.((ch \in out\ P) \wedge (exprChannel\ ch\ E) \wedge (ch \notin loc\ PQ))]\!]$$
$$\Rightarrow\ eout\ PQ\ E\text{”}$$

$$\text{“}[\![eoutM\ P\ M\ E;$$
$$subcomponents\ PQ = \{P, Q\};\ correctCompositionOut\ PQ;$$
$$\exists\, ch.((ch \in out\ Q) \wedge (exprChannel\ ch\ E) \wedge (ch \notin loc\ PQ) \wedge ch \in M)]\!]$$
$$\Rightarrow\ eoutM\ PQ\ M\ E\text{”}$$

"$[\![(eout\ \ P\ \ E) \vee (eout\ \ Q\ \ E);$
$subcomponents\ \ PQ = \{P, Q\};\ \ correctCompositionOut\ \ PQ;$
$\exists\, ch.((ch \in out\ \ P \vee ch \in out\ \ Q) \wedge (exprChannel\ \ ch\ \ E) \wedge (ch \notin loc\ \ PQ))]\!]$
$\Rightarrow\ \ eout\ \ PQ\ \ E$"

"$[\![(eoutM\ \ P\ \ M\ \ E) \vee (eoutM\ \ Q\ \ M\ \ E);$
$subcomponents\ \ PQ = \{P, Q\};\ \ correctCompositionOut\ \ PQ;$
$\exists\, ch.((ch \in out\ \ P \vee ch \in out\ \ Q) \wedge ch \in M \wedge (exprChannel\ \ ch\ \ E) \wedge (ch \notin loc\ \ PQ))]\!]$
$\Rightarrow\ \ eoutM\ \ PQ\ \ M\ \ E$"

$\Box$

Proposition 8 *For any components P and Q the following properties of the composition $P \otimes Q$ ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$) hold:*

$$P^{eout}(E) \wedge \neg Q^{eout}(E) \wedge \exists\, ch \in l_{P\otimes Q} : exprChannelSingleO(P, ch, E)$$
$$\rightarrow\ (P \otimes Q)^{eout}(E)$$

$$P^{eout}_M(E) \wedge \neg Q^{eout}_M(E) \wedge \exists\, ch \in l_{P\otimes Q} : (exprChannelSetO(P, ch, E) \wedge ch \in M)$$
$$\rightarrow\ (P \otimes Q)^{eout}_M(E)$$

The corresponding representation in Isabelle:

"$[\![eout\ \ P\ \ E;\ \neg(eout\ \ Q\ \ E);$
$subcomponents\ \ PQ = \{P, Q\};\ \ correctCompositionOut\ \ PQ;$
$\exists\, ch.((out\_exprChannelSingle\ \ P\ \ ch\ \ E) \wedge (ch \in loc\ \ PQ))]\!]$
$\Rightarrow\ \ \neg(eout\ \ PQ\ \ E)$"

"$[\![eoutM\ \ P\ \ M\ \ E;\ \neg(eoutM\ \ Q\ \ M\ \ E);$
$subcomponents\ \ PQ = \{P, Q\};\ \ correctCompositionOut\ \ PQ;$
$\exists\, ch.((out\_exprChannelSingle\ \ P\ \ ch\ \ E) \wedge ch \in M \wedge (ch \in loc\ \ PQ))]\!]$
$\Rightarrow\ \ \neg(eoutM\ \ PQ\ \ M\ \ E)$"

$\Box$

Proposition 9 *For any components P and Q the following properties of the composition $P \otimes Q$ ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$) hold:*

$$\neg Q^{eout}(E) \wedge exprChannelSetO(P, ChSet, E) \wedge \forall\, ch : ch \in ChSet \rightarrow ch \in l_{P\otimes Q}$$
$$\rightarrow\ \neg(P \otimes Q)^{eout}(E)$$

$$\neg Q^{\textsf{eout}}_M(E) \wedge exprChannelSetO(P, ChSet, E) \wedge \forall\, ch : ch \in ChSet \to ch \in l_{P \otimes Q}$$
$$\to\ \neg(P \otimes Q)^{\textsf{eout}}_M(E)$$

The corresponding representation in Isabelle:

"$[\![\neg(eout\ \ Q\ \ E);\ out\_exprChannelSet\ \ P\ \ ChSet\ \ E;$
$subcomponents\ \ PQ = \{P, Q\};\ correctCompositionOut\ \ PQ;$
$\forall(x :: chanID).(x \in ChSet \to x \in loc\ \ PQ)]\!]$
$\Rightarrow\ \neg(eout\ \ PQ\ \ E)$"

"$[\![\neg(eoutM\ \ Q\ \ M\ \ E);\ out\_exprChannelSet\ \ P\ \ ChSet\ \ E;$
$subcomponents\ \ PQ = \{P, Q\};\ correctCompositionOut\ \ PQ;$
$\forall(x :: chanID).(x \in ChSet \to (x \in loc\ \ PQ))]\!]$
$\Rightarrow\ \neg(eoutM\ \ PQ\ \ M\ \ E)$"

□

Proposition 10 *For any components P and Q the following properties of the composition $P \otimes Q$ ($e \in Expression$, $m \in KS$, $m \notin KS_P$ and $m \notin KS_Q$) hold:*

$$exprChannelSetO(P, ChSetP, E) \wedge exprChannelSetO(Q, ChSetQ, E) \wedge$$
$$\forall\, ch : ch \in ChSetP \to ch \in l_{P \otimes Q}\ \ \wedge\ \ \forall\, ch : ch \in ChSetQ \to ch \in l_{P \otimes Q}$$
$$\to\ \neg(P \otimes Q)^{\textsf{eout}}(E)$$

$$exprChannelSetO(P, ChSetP, E) \wedge exprChannelSetO(Q, ChSetQ, E) \wedge$$
$$M = ChSetP \cup ChSetQ\ \wedge$$
$$\forall\, ch : ch \in ChSetP \to ch \in l_{P \otimes Q}\ \ \wedge\ \ \forall\, ch : ch \in ChSetQ \to ch \in l_{P \otimes Q}$$
$$\to\ \neg(P \otimes Q)^{\textsf{eout}}_M(E)$$

The corresponding representation in Isabelle:

"$[\![subcomponents\ \ PQ = \{P, Q\};\ correctCompositionOut\ \ PQ;$
$out\_exprChannelSet\ \ P\ \ ChSetP\ \ E;\ out\_exprChannelSet\ \ Q\ \ ChSetQ\ \ E;$
$\forall(x :: chanID).(x \in ChSetP \to (x \in loc\ \ PQ));$
$\forall(x :: chanID).(x \in ChSetQ \to (x \in loc\ \ PQ))]\!]$
$\Rightarrow\ \neg(eout\ \ PQ\ \ E)$"

$$
\begin{array}{l}
\text{“}[\![subcomponents\ \ PQ = \{P, Q\};\ correctCompositionOut\ \ PQ;\\
\ out\_exprChannelSet\ \ P\ \ ChSetP\ \ E;\ out\_exprChannelSet\ \ Q\ \ ChSetQ\ \ E;\\
\ M = ChSetP \cup ChSetQ;\\
\ \forall (x :: chanID).(x \in ChSetP \to (x \in loc\ \ PQ));\\
\ \forall (x :: chanID).(x \in ChSetQ \to (x \in loc\ \ PQ))]\!]\\
\quad \Rightarrow\ \neg(eoutM\ \ PQ\ \ M\ \ E)\text{”}
\end{array}
$$

□

3.7 Set of Local Secrets

In addition to the sets of private keys and unguessable values of a component A we present in Isabelle/HOL according to the definition from [SJ08] 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:

consts

$LocalSecrets :: \text{“}specID \Rightarrow KSset\text{”}$

axioms

$LocalSecretsDef$:

$$
\begin{array}{l}
\text{“}LocalSecrets\ \ A =\\
\{(m :: KS).m \notin specKeysSecretsA \wedge\\
\quad ((\exists\, xy.(x \in loc\ \ A \wedge m = (kKS\ \ y) \wedge (exprChannel\ \ x\ \ (kE\ \ y))))\\
\quad \vee (\exists\, xz.(x \in loc\ \ A \wedge m = (sKS\ \ z) \wedge (exprChannel\ \ x\ \ (sE\ \ z)))))\}\\
\cup\\
(\bigcup (LocalSecrets\ \text{`}\ (subcomponents\ \ A)))\text{”}
\end{array}
$$

We defined a number of Isabelle/HOL lemmas describing properties of the set of the local secrets:

- If *ls* belongs to the set of local secrets of a subcomponent of the composite component $PQ = P \otimes Q$, then *ls* lelongs also to the set of local secrets of PQ.

  $$
  \begin{array}{l}
  \text{”}[\![ls \in LocalSecrets\ \ P;\ subcomponents\ \ PQ = \{P, Q\}]\!]\\
  \quad \Rightarrow ls \in LocalSecrets\ \ PQ\text{”}
  \end{array}
  $$

- If a key does not belong to the set of local secrets of any subcomponent of the composite component $PQ = P \otimes Q$ as well as cannot be eventually input by any of the subcomponents, then it also does not belong to the set of local secrets of PQ.

$$\text{“}[\![\mathit{subcomponents}\ PQ = \{P, Q\};\ \mathit{correctCompositionLoc}\ PQ;\ \neg \mathit{ine}\ P\ (\mathit{kE}\ \mathit{Keys});\ \mathit{kKS}\ \mathit{Keys} \notin \mathit{LocalSecretsP};\ \neg \mathit{ine}\ Q\ (\mathit{kE}\ \mathit{Keys});\ \mathit{kKS}\ \mathit{Keys} \notin \mathit{LocalSecrets}\ Q]\!]\ \Rightarrow \mathit{kKS}\ \mathit{Keys} \notin \mathit{LocalSecrets}\ PQ\text{”}$$

$$\text{“}[\![\mathit{subcomponents}\ PQ = \{P, Q\};\ \mathit{correctCompositionLoc}\ PQ;\ \mathit{correctCompositionKS}\ PQ;\ (\mathit{kKS}\ m) \notin \mathit{specKeysSecrets}\ P;\ (\mathit{kKS}\ m) \notin \mathit{specKeysSecrets}\ Q;\ \neg(\mathit{ine}\ P\ (\mathit{kE}\ m));\ \neg(\mathit{ine}\ Q\ (\mathit{kE}\ m));\ (\mathit{kKS}\ m) \notin ((\mathit{LocalSecrets}\ P) \cup (\mathit{LocalSecrets}\ Q))]\!]\ \Rightarrow (\mathit{kKS}\ m) \notin (\mathit{LocalSecrets}\ PQ)\text{”}$$

- If an unguessable value does not belong to the set of local secrets of any subcomponent of the composite component $PQ = P \otimes Q$ as well as cannot be eventually input by any of the subcomponents, then it also does not belong to the set of unguessable values of PQ.

$$\text{“}[\![\mathit{subcomponents}\ PQ = \{P, Q\};\ \mathit{correctCompositionLoc}\ PQ;\ \neg \mathit{ine}\ P\ (\mathit{sE}\ s);\ \mathit{sKS}\ s \notin \mathit{LocalSecrets}\ P;\ \neg \mathit{ine}\ Q\ (\mathit{sE}\ s);\ \mathit{sKS}\ s \notin \mathit{LocalSecrets}\ Q]\!]\ \Rightarrow \mathit{sKS}\ s \notin \mathit{LocalSecrets}\ PQ\text{”}$$

$$\text{“}[\![\mathit{subcomponents}\ PQ = \{P, Q\};\ \mathit{correctCompositionLoc}\ PQ;\ \mathit{correctCompositionKS}\ PQ;\ (\mathit{sKS}\ m) \notin \mathit{specKeysSecrets}\ P;\ (\mathit{sKS}\ m) \notin \mathit{specKeysSecrets}\ Q;\ \neg(\mathit{ine}\ P\ (\mathit{sE}\ m));\ \neg(\mathit{ine}\ Q\ (\mathit{sE}\ m));\ (\mathit{sKS}\ m) \notin ((\mathit{LocalSecrets}\ P) \cup (\mathit{LocalSecrets}\ Q))]\!]\ \Rightarrow (\mathit{sKS}\ m) \notin (\mathit{LocalSecrets}\ PQ)\text{”}$$

- If a key or an unguessable value does not belong to the set of local secrets of any subcomponent of the composite component $PQ = P \otimes Q$ as well as cannot be eventually input by any of the subcomponents, then it also does not belong to the set of keys and unguessable values of PQ.

$$\text{“}[\![subcomponents\ PQ = \{P, Q\};\ correctCompositionLoc\ PQ;$$
$$\forall\, m.\ ks = kKSm \rightarrow (\neg(ine\ P\ (kE\ m)) \wedge \neg(ine\ Q\ (kE\ m)));$$
$$\forall\, m.\ ks = sKSm \rightarrow (\neg(ine\ P\ (sE\ m)) \wedge \neg(ine\ Q\ (sE\ m)));$$
$$ks \notin LocalSecrets\ P;\ ks \notin LocalSecrets\ Q]\!]$$
$$\Rightarrow ks \notin LocalSecrets\ PQ\text{”}$$

$$\text{“}[\![subcomponents\ PQ = \{P, Q\};$$
$$correctCompositionLoc\ PQ;\ correctCompositionKS\ PQ;$$
$$ks \notin specKeysSecrets\ P;\ ks \notin specKeysSecrets\ Q;$$
$$\forall\, m.\ ks = kKSm \rightarrow (\neg(ine\ P\ (kE\ m)) \wedge\ (ine\ Q\ (kE\ m)));$$
$$\forall\, m.\ ks = sKSm \rightarrow (\neg(ine\ P\ (sE\ m)) \wedge\ (ine\ Q\ (sE\ m)));$$
$$ks \notin ((LocalSecretsP)\, Un(LocalSecretsQ))]\!]$$
$$\Rightarrow ks \notin (LocalSecrets\ PQ)\text{”}$$

- If a key belongs to the set of keys of the composite component $PQ = P \otimes Q$, but does not belong to the set of keys and unguessable values of P and Q, as well as cannot be eventually input by PQ and as cannot be eventually input by its subcomponent Q, then it must be eventually input by P.

$$\text{“}[\![kKS\ k \in LocalSecrets\ PQ;$$
$$subcomponents\ PQ = \{P, Q\};\ correctCompositionLoc\ PQ;$$
$$\neg ine\ PQ\ (kE\ k);\ \neg ine\ Q\ (kE\ k);$$
$$kKS\ k \notin LocalSecrets\ P;\ kKS\ k \notin LocalSecrets\ Q]\!]$$
$$\Rightarrow ine\ P\ (kE\ k)\text{”}$$

- If an unguessable value belongs to the set of unguessable values of the composite component $PQ = P \otimes Q$, but does not belong to the set of keys and unguessable values of P and Q, as well as cannot be eventually input by PQ and as cannot be eventually input by its subcomponent Q, then it must be eventually input by P.

$$\text{“}[\![sKS\ s \in LocalSecrets\ PQ;$$
$$subcomponents\ PQ = \{P, Q\};\ correctCompositionLoc\ PQ;$$
$$\neg ine\ PQ\ (sE\ s);\ \neg ine\ Q\ (sE\ s);$$
$$sKSs \notin LocalSecrets\ P;\ sKS\ s \notin LocalSecrets\ Q]\!]$$
$$\Rightarrow ine\ P\ (sE\ s)\text{”}$$

- If a key belongs to the set of keys of the composite component $PQ = P \otimes Q$, but does not belong to the set of keys and unguessable values of P and Q, as well as cannot be eventually input by PQ and as cannot be eventually input by its subcomponent P, then it must be eventually input by Q.

$$\text{“}[\![kKS\ k \in LocalSecrets\ PQ;\ subcomponents\ PQ = \{P, Q\};\ correctCompositionLoc\ PQ;\ \neg ine\ PQ\ (kE\ k);\ \neg ine\ P\ (kE\ k);\ kKS\ k \notin LocalSecrets\ P;\ kKS\ k \notin LocalSecrets\ Q]\!]\ \Rightarrow ine\ Q\ (kE\ k)\text{”}$$

- If an unguessable value belongs to the set of unguessable values of the composite component $PQ = P \otimes Q$, but does not belong to the set of keys and unguessable values of P and Q, as well as cannot be eventually input by PQ and as cannot be eventually input by its subcomponent P, then it must be eventually input by Q.

$$\text{“}[\![sKS\ s \in LocalSecrets\ PQ;\ subcomponents\ PQ = \{P, Q\};\ correctCompositionLoc\ PQ;\ \neg ine\ PQ\ (sE\ s);\ \neg ine\ P\ (sE\ s);\ sKS\ s \notin LocalSecrets\ P;\ sKS\ s \notin LocalSecrets\ Q]\!]\ \Rightarrow ine\ Q\ (sE\ s)\text{”}$$

3.8 Knowledges of An Adversary

As presented in [SJ08], an (*adversary*) component A knows a secret $m \in KS$, $m \notin KS_A$ (or some secret expression m, $m \in (Expression \setminus KS_A)^*$), 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.

We represent these definition in Isabelle/HOL distinguishing (like in Focus) 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.:

$$\mathsf{know}^A \in KS \setminus KS_A \to \mathbb{B}\mathrm{ool}$$
$$\mathsf{know}^A(m) \stackrel{\mathsf{def}}{=} A^{\mathsf{ine}}(m) \ \lor\ m \in LS_A$$

consts

know :: "*specID* ⇒ *KS* ⇒ *bool*"

primrec

"*know A (kKS m) = ((ine A (kE m)) | ((kKS m) ∈ LocalSecrets A))*"

"*know A (sKS m) = ((ine A (sE m)) | ((sKS m) ∈ LocalSecrets A))*"

$$\mathsf{knows}^A \in (Expression \setminus KS_A)^* \to \mathbb{B}\mathrm{ool}$$
$$\mathsf{knows}^A(m) \stackrel{\mathsf{def}}{=}$$
$$(\exists\, m_1 : m = \langle m_1 \rangle \ \land\ \mathsf{know}^A(m_1)) \ \lor$$
$$(\exists\, m_1, m_2 : (m_2 = m \frown m_1 \ \lor\ m_2 = m_1 \frown m) \land \mathsf{knows}^A(m_2)) \ \lor$$
$$(\exists\, m_1, m_2 : m = m_1 \frown m_2 \ \land\ \mathsf{knows}^A(m_1) \ \land\ \mathsf{knows}^A(m_2)) \ \lor$$
$$(\exists\, k, k^{-1} : \mathsf{know}^A(k^{-1}) \ \land\ \mathsf{knows}^A(Enc(k, m))) \ \lor$$
$$(\exists\, k, k^{-1} : \mathsf{know}^A(k) \ \land\ \mathsf{knows}^A(Sign(k^{-1}, m))) \ \lor$$
$$(\exists\, k, m_1 : m = Enc(k, m_1) \ \land\ \mathsf{knows}^A(m_1) \ \land\ \mathsf{know}^A(k)) \ \lor$$
$$(\exists\, k, m_1 : m = Sign(k, m_1) \ \land\ \mathsf{knows}^A(m_1) \ \land\ \mathsf{know}^A(k))$$

consts

knows :: "*specID* ⇒ *Expressionlist* ⇒ *bool*"

axioms

knows1k :

"*know A (kKS m) = knows A [kE m]*"

know1k :

"*knows A [kE m] = know A (kKS m)*"

knows1s :

"*know A (sKS m) = knows A [sE m]*"

know1s :

"*knows A [sE m] = know A (sKS m)*"

$knows2$:

"$[\![\, \exists\, e1\ e2.\ (e2 = e1@e \vee e2 = e@e1) \wedge (knows\ A\ e2)]\!] \Rightarrow knows\ A\ e$"

$knows2a$:

"$[\![\, \exists\, e1\ e2.\ (e2 = e1@e) \wedge (knows\ A\ e2)]\!] \Rightarrow knows\ A\ e$"

$knows2b$:

"$[\![\, \exists\, e1\ e2.\ ((e2 = e@e1) \wedge (knows\ A\ e2))]\!] \Rightarrow knows\ A\ e$"

$knows3$:

"$[\![\, \exists\, e1\ e2.\ (e = e1@e2 \wedge (knows\ A\ e1) \wedge (knows\ A\ e2))]\!] \Rightarrow knows\ A\ e$"

$knows4$:

"$[\![\, \exists\, k1\ k2.\ ((IncrDecrKeys\ k1\ k2) \wedge (know\ A\ (kKS\ k2)) \wedge (knows\ A\ (Enc\ k1\ e)))]\!]$
$\Rightarrow knows\ A\ e$"

$knows5$:

"$[\![\, \exists\, k1k2.\ ((IncrDecrKeys\ k1\ k2) \wedge (know\ A\ (kKS\ k1)) \wedge (knows\ A\ (Sign\ k2\ e)))]\!]$
$\Rightarrow knows\ A\ e$"

$knows6$:

"$[\![\, \exists\, ke1.\ (e = (Enc\ k\ e1) \wedge (know\ A\ (kKS\ k)) \wedge (knows\ A\ e1))]\!] \Rightarrow knows\ A\ e$"

$knows7$:

"$[\![\, \exists\, ke1.\ (e = (Sign\ k\ e1) \wedge (know\ A\ (kKS\ k)) \wedge (knows\ A\ e1))]\!] \Rightarrow knows\ A\ e$"

We also add a number of axioms that describe relations between the predicates $know(s)$ and the predicate describing that a component may eventually output an expression.

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

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

The corresponding axioms in Isabelle:

axioms

$eout\_know\_k$:

"$\forall (C :: specID)(m :: Keys).$
$(eout\ \ C\ (kE\ m)) = ((m \in (specKeys\ C)) \mid (know\ C\ (kKS\ m)))$"

$eout\_know\_s$:

"$\forall (C :: specID)(s :: Secrets).$
$(eout\ \ C\ (sE\ s)) = ((s \in (specSecrets\ C)) \mid (know\ C\ (sKS\ s)))$"

$eout\_knows$:

"$\forall (C :: specID)(e :: Expression).$

$(eout\ \ C\ \ e) = ((\exists\, k.\ \ e = (kE\ \ k) \wedge (k \in specKeys\ \ C))$

$\mid (\exists\, s.\ \ e = (sE\ \ s) \wedge (s \in specSecrets\ \ C))$

$\mid (knows\ \ C\ \ [e]))$"

□

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

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

The corresponding axiom in Isabelle:

axioms

$knows\_emptyexpression$:

"$knows\ \ C\ \ []$"

□

We omit here a number of lemmas to concentrate on more important ones from our point of view. For the whole collection of lemmas we would like to refer to the Isabelle/HOL theory *AdvKnowledge.thy*.

Proposition 11 *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'*):*

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

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

The corresponding lemmas in Isabelle:

"$[\![m \notin specKeys\ \ A;\ \ eout\ \ A\ \ (kE\ \ m)]\!] \Rightarrow know\ \ A\ \ (kKS\ \ m)$"

"$[\![m \notin specSecrets\ \ A;\ \ eout\ \ A\ \ (sE\ \ m)]\!] \Rightarrow know\ \ A\ \ (sKS\ \ m)$"

"$[\![m \notin (specKeys\ \ A);\ \ eout\ \ A\ \ (kE\ \ m)]\!] \Rightarrow knows\ \ A\ \ [kE\ \ m]$"

"$[\![m \notin specSecrets\ \ A;\ \ eout\ \ A\ \ (sE\ \ m)]\!] \Rightarrow knows\ \ A\ \ [sE\ \ m]$"

□

Proposition 12 *If an* adversary *component* A *does not know a secret or a key* $m \in KS$*, then this component* A *cannot eventually get* m*:*

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

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

The corresponding lemmas in Isabelle:

"$\neg know\ A\ (kKS\ m) \to \neg ine\ A\ (kE\ m)$"
"$\neg know\ A\ (sKS\ m) \to \neg ine\ A\ (sE\ m)$"
"$[\![\neg knows\ A\ [kE\ m]]\!] \Rightarrow \neg ine\ A\ (kE\ m)$"
"$\neg knows\ A\ [sE\ m] \Rightarrow \neg ine\ A\ (sE\ m)$"

$\square$

Proposition 13 *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:*

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

The corresponding lemmas in Isabelle:

"$[\![m \notin specKeys\ A;\ \neg know\ A\ (kKS\ m)]\!] \Rightarrow \neg eout\ A\ (kE\ m)$"
"$[\![m \notin specSecrets\ A;\ \neg know\ A\ (sKS\ m)]\!] \Rightarrow \neg eout\ A\ (sE\ m)$"
"$[\![m \notin specKeys\ A;\ \neg knows\ A\ [kE\ m]]\!] \Rightarrow \neg eout\ A\ (kE\ m)$"
"$[\![m \notin specSecrets\ A;\ \neg knows\ A\ [sE\ m]]\!] \Rightarrow \neg eout\ A\ (sE\ m)$"

$\square$

Proposition 14 *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:*

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

The corresponding lemmas in Isabelle:

"$[\![out\ P \subseteq ins\ A;\ \neg know\ A\ (kKS\ m)]\!] \Rightarrow \neg eout\ P\ (kE\ m)$"
"$[\![out\ P \subseteq ins\ A;\ \neg know\ A\ (sKS\ m)]\!] \Rightarrow \neg eout\ P\ (sE\ m)$"
"$[\![out\ P \subseteq ins\ A;\ \neg knows\ A\ [kE\ m]]\!] \Rightarrow \neg eout\ P\ (kE\ m)$"
"$[\![out\ P \subseteq ins\ A;\ \neg knows\ A\ [sE\ m]]\!] \Rightarrow \neg eout\ P\ (sE\ m)$"

$\square$

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 $m \notin KS_Q$):*

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

The corresponding lemma in Isabelle:

"$[\![m \notin specKeysSecrets\ P;\ m \notin specKeysSecrets\ Q;$
$know\ P\ m;\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Rightarrow know\ PQ\ m$"

□

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 $m \notin KS_Q$):*

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

The corresponding lemma in Isabelle:

"$[\![m \notin specKeysSecrets\ P;\ m \notin specKeysSecrets\ Q;$
$know\ Q\ m;\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Rightarrow know\ PQ\ m$"

□

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 $m \notin KS_Q$):*

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

The corresponding lemma in Isabelle:

"$[\![m \notin specKeysSecrets\ P;\ m \notin specKeysSecrets\ Q;$
$(know\ P\ m \vee know\ Q\ m);\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Rightarrow know\ PQ\ m$"

□

Theorem 9 *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:*

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

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

The corresponding lemmas in Isabelle:

"$[\![m \notin specKeysSecrets\ P;\ m \notin specKeysSecrets\ Q;$
$\neg know\ P\ m;\ \neg know\ Q\ m;$
$subcomponents\ PQ = \{P, Q\};\ correctCompositionLoc\ PQ;$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Longrightarrow \neg know\ PQ\ m$"

"$[\![m \notin specKeysSecrets\ P;\ m \notin specKeysSecrets\ Q;$
$know\ PQ\ m;\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionLoc\ PQ]\!]$
$\Longrightarrow know\ P\ m \vee know\ Q\ m$"

□

Proposition 15 *For any components P and Q the composition $P \otimes Q$ has the following properties ($e \in KS^*$):*

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

The corresponding lemmas in Isabelle:

"$[\![(kKS\ m) \notin specKeysSecrets\ P;\ (kKS\ m) \notin specKeysSecrets\ Q;$
$knows\ P\ [kE\ m];\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Longrightarrow knows\ PQ\ [kE\ m]$"

"$[\![(sKS\ m) \notin specKeysSecrets\ P;\ (sKS\ m) \notin specKeysSecrets\ Q;$
$knows\ P\ [sE\ m];\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Longrightarrow knows\ PQ\ [sE\ m]$"

"$[\![(kKS\ m) \notin specKeysSecrets\ P;\ (kKS\ m) \notin specKeysSecrets\ Q;$
$knows\ Q\ [kE\ m];\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Longrightarrow knows\ PQ\ [kE\ m]$"

$$“\llbracket (sKS\ \ m) \notin specKeysSecrets\ \ P;\ (sKS\ \ m) \notin specKeysSecrets\ \ Q;$$
$$knows\ \ Q\ [sE\ \ m];\ subcomponents\ \ PQ = \{P, Q\};$$
$$correctCompositionIn\ \ PQ;\ correctCompositionKS\ \ PQ\rrbracket$$
$$\Rightarrow knows\ \ PQ\ [sE\ \ m]”$$

□

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

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

The corresponding lemmas in Isabelle:

$$“\llbracket kKS\ \ a \notin specKeysSecrets\ \ P;\ kKS\ \ a \notin specKeysSecrets\ \ Q;$$
$$subcomponents\ \ PQ = \{P, Q\};\ knows\ \ P\ [kE\ \ a];$$
$$correctCompositionIn\ \ PQ;\ correctCompositionKS\ \ PQ\rrbracket$$
$$\Rightarrow knows\ \ PQ\ [kE\ \ a]”$$

$$“\llbracket sKS\ \ a \notin specKeysSecrets\ \ P;\ sKS\ \ a \notin specKeysSecrets\ \ Q;$$
$$subcomponents\ \ PQ = \{P, Q\};\ knows\ \ P\ [sE\ \ a];$$
$$correctCompositionIn\ \ PQ;\ correctCompositionKS\ \ PQ\rrbracket$$
$$\Rightarrow knows\ \ PQ\ [sE\ \ a]”$$

$$“\llbracket knows\ \ P\ \ e;\ subcomponents\ \ PQ = \{P, Q\};$$
$$correctCompositionIn\ \ PQ;\ correctCompositionKS\ \ PQ;$$
$$\forall m.\ m\ \ mem\ \ e \rightarrow ((\exists z.\ m = kEz) \mid (\exists z.\ m = sEz));$$
$$\forall x.\ (kE\ \ x)\ \ mem\ \ e \rightarrow (kKSx) \notin specKeysSecrets\ \ P;$$
$$\forall y.\ (sE\ \ y)\ \ mem\ \ e \rightarrow (sKSy) \notin specKeysSecrets\ \ P;$$
$$\forall x.\ (kE\ \ x)\ \ mem\ \ e \rightarrow kKSx \notin specKeysSecrets\ \ Q;$$
$$\forall y.\ (sE\ \ y)\ \ mem\ \ e \rightarrow sKSy \notin specKeysSecrets\ \ Q\rrbracket$$
$$\Rightarrow knows\ \ PQ\ \ e”$$

□

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

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

The corresponding lemmas in Isabelle:

"$[\![kKS\ a \notin specKeysSecrets\ P;\ kKSa \notin specKeysSecrets\ Q;$
$subcomponents\ PQ = \{P, Q\};\ knows\ Q\ [kE\ a];$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Rightarrow knows\ PQ\ [kE\ a]$"

"$[\![sKS\ a \notin specKeysSecrets\ P;\ sKS\ a \notin specKeysSecrets\ Q;$
$subcomponents\ PQ = \{P, Q\};\ knows\ Q\ [sE\ a];$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Rightarrow knows\ PQ\ [sE\ a]$"

"$[\![knows\ Q\ e;\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ;$
$\forall m.\ m\ mem\ e \longrightarrow ((\exists z.\ m = kEz) \mid (\exists z.\ m = sEz));$
$\forall x.\ (kE\ x)\ mem\ e \longrightarrow (kKSx) \notin specKeysSecrets\ P;$
$\forall y.\ (sE\ y)\ mem\ e \longrightarrow (sKSy) \notin specKeysSecrets\ P;$
$\forall x.\ (kE\ x)\ mem\ e \longrightarrow kKSx \notin specKeysSecrets\ Q;$
$\forall y.\ (sE\ y)\ mem\ e \longrightarrow sKSy \notin specKeysSecrets\ Q]\!]$
$\Rightarrow knows\ PQ\ e$"

□

Theorem 12 *For any components P and Q the composition $P \otimes Q$ has the following property ($e \in KS^*$):*

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

The corresponding lemma in Isabelle:

"$[\![knows\ P\ e \vee knows\ Q\ e;\ subcomponents\ PQ = \{P, Q\};$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ;$
$\forall m.\ m\ mem\ e \longrightarrow ((\exists z.\ m = kEz) \mid (\exists z.\ m = sEz));$
$\forall x.\ (kE\ x)\ mem\ e \longrightarrow (kKSx) \notin specKeysSecrets\ P;$
$\forall y.\ (sE\ y)\ mem\ e \longrightarrow (sKSy) \notin specKeysSecrets\ P;$
$\forall x.\ (kE\ x)\ mem\ e \longrightarrow kKSx \notin specKeysSecrets\ Q;$
$\forall y.\ (sE\ y)\ mem\ e \longrightarrow sKSy \notin specKeysSecrets\ Q]\!]$
$\Rightarrow knows\ PQ\ e$"

□

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:*

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

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

The corresponding lemmas in Isabelle:

"$[\![kKS\ m \notin specKeysSecrets\ P;\ kKS\ m \notin specKeysSecrets\ Q;$
$\neg knows\ P\ [kE\ m];\ \neg knows\ Q\ [kE\ m];$
$subcomponents\ PQ = \{P, Q\};\ correctCompositionLoc\ PQ;$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Rightarrow \neg knows\ PQ\ [kE\ m]$"

"$[\![sKS\ m \notin specKeysSecrets\ P;\ sKS\ m \notin specKeysSecrets\ Q;$
$\neg knows\ P\ [sE\ m];\ \neg knows\ Q\ [sE\ m];$
$subcomponents\ PQ = \{P, Q\};\ correctCompositionLoc\ PQ;$
$correctCompositionIn\ PQ;\ correctCompositionKS\ PQ]\!]$
$\Rightarrow \neg knows\ PQ\ [sE\ m]$"

"$[\![kKS\ a \notin specKeysSecrets\ P;\ kKS\ a \notin specKeysSecrets\ Q;$
$subcomponents\ PQ = \{P, Q\};\ knows\ PQ\ [kE\ a];$
$correctCompositionIn\ PQ;\ correctCompositionLoc\ PQ]\!]$
$\Rightarrow knows\ P\ [kE\ a] \vee knows\ Q\ [kE\ a]$"

"$[\![sKS\ a \notin specKeysSecrets\ P;\ sKS\ a \notin specKeysSecrets\ Q;$
$subcomponents\ PQ = \{P, Q\};\ knows\ PQ\ [sE\ a];$
$correctCompositionIn\ PQ;\ correctCompositionLoc\ PQ]\!]$
$\Rightarrow knows\ P\ [sE\ a] \vee knows\ Q\ [sE\ a]$"

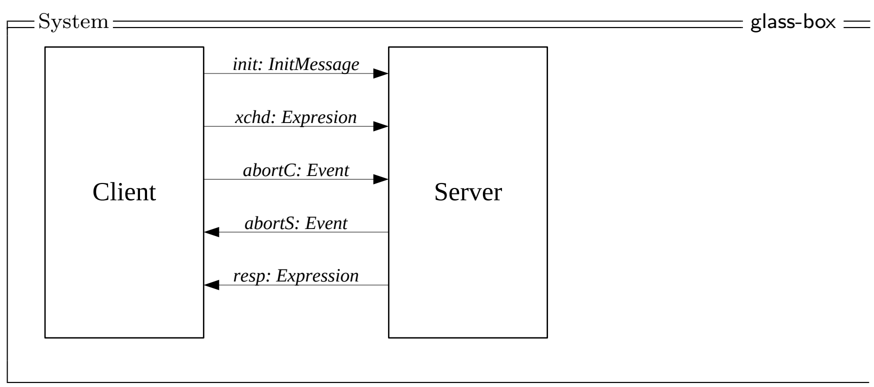
"$[\![subcomponents\ PQ = \{P, Q\};\ knows\ PQ\ [a];$
$correctCompositionIn\ PQ;\ correctCompositionLoc\ PQ;$
$(\exists\, z.\ a = kE\ z) \vee (\exists\, z.\ a = sE\ z);$
$\forall\, z.\ a = kE\ z \to kKS\ z \notin specKeysSecrets\ P \wedge kKS\ z \notin specKeysSecrets\ Q;$
$\forall\, z.\ a = sE\ z \to sKS\ z \notin specKeysSecrets\ P \wedge sKS\ z \notin specKeysSecrets\ Q]\!]$
$\Rightarrow knowsP[a] \vee knowsQ[a]$"

□

4 TLS Protocol

To demonstrate usability of our approach, we specified in [SJ08] a variant of the handshake protocol of TLS[1] [APS99], which goal 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. Here we present the optimized version of the FOCUS specifications of the protocol components as well as their translation to Isabelle/HOL and the corresponding lemmas that show the most important properties of the protocol.

Let us recall the general idea of the handshake protocol of TLS. The protocol has two participants, *Client* and *Server*, that are connected by an Internet connection. We used the following auxiliary data types: $Obj = \{C, S\}$, $StateC = \{st0, st1, st2\}$ and $StateS = \{initS, waitS, sendS1, sendS2\}$ to represent participants names and states, $Event = \{event\}$ to represent message sending events (e.g. an abort message or an acknowledgment), and $InitMessage = im(ungValue \in Secret, key \in Keys,\ msg \in Expression)$ to represent events initiating the protocol by the client.



4.1 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 $CKey^{-1}$.

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.

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

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.

Below we present the optimized FOCUS specifications of these components. In comparison with the specification we shown in [SJ08], the new version is more readable and uses local variables to allow more clear presentation of the data exchange sequence.

Client — timed

in $abortS : Event;\ resp : Expression$

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

local $check : StateC;\ enc : Keys$

init $check = st0;$

asm true

gar

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

2 $\mathsf{ti}(xchd, 0) = \langle\rangle$

3 $\mathsf{ti}(abortC, 0) = \langle\rangle$

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

4 $\mathsf{ti}(init, t+1) = \langle\rangle$

5 $\mathsf{ti}(abortS, t) \neq \langle\rangle \rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t+1) = \langle\rangle \wedge check' = st0$

6 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) = \langle\rangle \wedge check = st0$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = st0$

7 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check' = st0$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = st1$

8 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check = st1 \wedge \mathsf{ft}.secr = S$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = st2 \wedge enc' = \mathsf{snd}.secr$

9 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check = st2 \wedge \mathsf{snd}.res = N$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t+1) = Enc(\mathsf{ft}.res, secretD) \wedge check' = st0$

10 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge$
$((check = st1 \wedge (\mathsf{ti}(resp, t) = \langle\rangle \vee (\mathsf{ti}(resp, t) \neq \langle\rangle \wedge \mathsf{ft}.secr \neq S))) \vee$
$(check = st2 \wedge (\mathsf{ti}(resp, t) = \langle\rangle \vee (\mathsf{ti}(resp, t) \neq \langle\rangle \wedge \mathsf{snd}.res \neq N))))$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle event\rangle \wedge$
$\mathsf{ti}(xchd, t+1) = \langle\rangle \wedge$
$check' = st0$

where
$secr = Ext(CAKey, \mathsf{ti}(resp, t))$
$res = Ext(enc, Decr(CKey^{-1}, \mathsf{ti}(resp, t)))$

Please note that we omit in this Focus specification of the server (like in the specification from [SJ08]) all the information how the server deals with the data it gets via the $xchd$ channel.

Server — timed

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

out $resp : Expression;\ abortS : Event$

local $stateS \in StateS;\ kValue \in Keys;\ uValue \in Secret$

init $stateS = initS$

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

gar

1 $\mathsf{ti}(resp, 0) = \langle\rangle$

2 $\mathsf{ti}(abortS, 0) = \langle\rangle$

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

3 $\mathsf{ti}(abortC, t) \neq \langle\rangle$
$\to stateS' = initS \ \wedge\ \mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

4 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = waitS$
$\to \mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ stateS' = waitS \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

5 $\mathsf{ti}(abortC, t) = \langle\rangle \wedge stateS = initS \wedge \mathsf{ti}(init, t) = \langle\rangle$
$\to \mathsf{ti}(resp, t+1) = \langle\rangle \wedge stateS' = initS \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

6 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = initS \ \wedge\ \mathsf{ti}(init, t) \neq \langle\rangle \ \wedge$
$\mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) \neq key(init^t_{\mathsf{ft}})$
$\to \mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ stateS' = initS \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle event\rangle$

7 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = initS \ \wedge\ \mathsf{ti}(init, t) \neq \langle\rangle \ \wedge$
$\mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) = key(init^t_{\mathsf{ft}})$
$\to \mathsf{ti}(resp, t+1) = \langle ungValue(init^t_{\mathsf{ft}})\rangle$
$\wedge\ stateS' = sendS1 \ \wedge\ uValue' = ungValue(init^t_{\mathsf{ft}}) \ \wedge\ kValue' = key(init^t_{\mathsf{ft}})$
$\wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

8 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = sendS1$
$\to \mathsf{ti}(resp, t+1) = Sign(CAKey^{-1}, \langle S, SKey\rangle)$
$\wedge\ stateS' = sendS2 \ \wedge\ uValue' = uValue \ \wedge\ kValue' = kValue$
$\wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

9 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = sendS2$
$\to \mathsf{ti}(resp, t+1) = Enc(kValue, Sign(SKey^{-1}, \langle genKey, uValue\rangle))$
$\wedge\ stateS' = waitS \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

The corresponding Isabelle/HOL representation of these two components looks like follows:

constdefs

Client_L ::
"*Event istream* ⇒ *Expression istream* ⇒ *StateC iustream* ⇒ *Keys iustream* ⇒
initMessage istream ⇒ *Expression istream* ⇒ *Event istream* ⇒ *StateC iustream* ⇒ *Keys iustream*
⇒ *bool*"

"*Client_L abortS resp check enc init xchd abortC checkNext encNext*
≡
(*True*
→
((*init* (0 :: *nat*)) = [(| *ungValue* = *N*, *key* = *CKey*, *imsg* = (*Sign CKeyP* [*idE sClient*, *kE CKey*]) |)] ∧
(*xchd* (0 :: *nat*)) = [] ∧
(*abortC* (0 :: *nat*) = [])
∧
(∀(*t* :: *nat*).(
init (*Suc t*) = []
∧
(*abortS t* ≠ []
→ (*xchd* (*Suc t*) = []) ∧ (*abortC* (*Suc t*) = []) ∧ *checkNext t* = *st*0)
∧
((*abortS t* = []) ∧ (*resp t* = []) ∧ (*check t* = *st*0)
→ (*xchd* (*Suc t*) = []) ∧ (*abortC* (*Suc t*) = []) ∧ *checkNext t* = *st*0)
∧
((*abortS t* = []) ∧ (*resp t* ≠ []) ∧ (*check t* = *st*0)
→ (*abortC* (*Suc t*) = []) ∧ (*xchd* (*Suc t*) = []) ∧ *checkNext t* = *st*1)
∧
((*abortS t* = []) ∧ (*resp t* ≠ []) ∧ *check t* = *st*1 ∧
(*hd* (*Ext CAKey* (*resp t*))) = *idE sServer* → (*xchd* (*Suc t*) = []) ∧ (*abortC* (*Suc t*) = [])
∧ *checkNext t* = *st*2
∧ *kE* (*encNext t*) = *hd* (*tl* (*Ext CAKey* (*resp t*))))
∧
((*abortS t* = []) ∧ (*resp t* ≠ []) ∧ (*check t* = *st*2) ∧
(*sE N* = *hd* (*tl* (*Ext* (*enc t*)(*Decr CKeyP* (*resp t*)))))
→ (*xchd* (*Suc t*) = *Enc* (*Expr*2*Keys* (*hd* (*Ext* (*enc t*)(*Decr CKeyP* (*resp t*))))) [*sE secretD*])
∧ (*abortC* (*Suc t*) = []) ∧ *checkNext t* = *st*0)
∧
((*abortS t* = []) ∧ (*check t* = *st*1)∧
((*resp t* = [])∨
((*resp t* ≠ []) ∧ (*hd* (*Ext CAKey* (*resp t*))) ≠ *idE sServer*))
→ (*xchd* (*Suc t*) = []) ∧ (*abortC* (*Suc t*) = [*event*])
∧*checkNext t* = *st*0)
∧
((*abortS t* = []) ∧ (*check t* = *st*2) ∧
((*resp t* = [])∨
((*resp t* ≠ []) ∧ (*sE N* ≠ *hd* (*tl* (*Ext* (*enc t*) (*Decr CKeyP* (*resp t*)))))))
→ (*xchd* (*Suc t*) = []) ∧ (*abortC* (*Suc t*) = [*event*]) ∧ *checkNext t* = *st*0)))))"

constdefs

Client :: "*Event istream* ⇒ *Expression istream* ⇒
initMessage istream ⇒ *Expression istream* ⇒ *Event istream* ⇒ *bool*"

"*Client abortS resp init xchd abortC* ≡
(∃ *check enc*.
Client_L abortS resp (*fin_inf_append* [*st*0] *check*) (*fin_inf_append* [*CKey*] *enc*) *init xchd abortC check enc*)"

constdefs
Server_L ::
“*InitMessage istream* ⇒ *Event istream* ⇒ *Expression istream* ⇒
StateS iustream ⇒ *Keys iustream* ⇒ *Secrets iustream* ⇒
Expression istream ⇒ *Event istream*
⇒ *StateS iustream* ⇒ *Keys iustream* ⇒ *Secrets iustream* ⇒ *bool*”

“*Server_L init abortC xchd stateS kValue uValue*
resp abortS stateSNext kValueNext uValueNext
≡
(((*msg* (1 :: *nat*) *init*) ∧ (*msg* (1 :: *nat*) *xchd*)) → (*resp* (0 :: *nat*) = []
∧
abortS (0 :: *nat*) = []
∧
(∀(*t* :: *nat*).(
((*abortC t* ≠ [])
→ *stateSNext t* = *initS* ∧ *resp* (*Suc t*) = [] ∧ *abortS* (*Suc t*) = [])
∧
((*abortC t* = []) ∧ *stateS t* = *waitS*
→ *stateSNext t* = *waitS* ∧ *resp* (*Suc t*) = [] ∧ *abortS* (*Suc t*) = [])
∧
((*abortC t* = []) ∧ *stateS t* = *initS* ∧ *init t* = []
→ *stateSNext t* = *initS* ∧ *resp* (*Suc t*) = [] ∧ *abortS* (*Suc t*) = [])
∧
((*abortC t* = []) ∧ *stateS t* = *initS* ∧ *init t* ≠ []∧
(*Expr2Keys* (*hd* (*tl* (*Ext* (*key* (*hd* (*init t*))) (*imsg* (*hd* (*init t*))))))) ≠ *key* (*hd* (*init t*))
→ *stateSNext t* = *initS* ∧ *resp* (*Suc t*) = [] ∧ *abortS* (*Suc t*) = [*event*])
∧
((*abortC t* = []) ∧ *stateS t* = *initS* ∧ *init t* ≠ []∧
(*Expr2Keys* (*hd* (*tl* (*Ext* (*key* (*hd* (*init t*))) (*imsg* (*hd* (*init t*))))))) = *key* (*hd* (*init t*))
→ *stateSNext t* = *sendS*1
∧ *resp* (*Suc t*) = [*sE*(*ungValue*(*hd*(*init t*)))]
∧ *abortS* (*Suc t*) = []
∧ *uValueNext t* = *ungValue*(*hd*(*init t*)) ∧ *kValueNext t* = *key*(*hd*(*init t*)))
∧
((*abortC t* = []) ∧ *stateS t* = *sendS*1
→ *stateSNext t* = *sendS*2
∧ *resp* (*Suc t*) = *Sign CAKeyP* [*idE sServer*, *kE SKey*]
∧ *abortS* (*Suc t*) = [] ∧ *uValueNext t* = *uValue t* ∧ *kValueNext t* = *kValue t*)
∧
((*abortC t* = []) ∧ *stateS t* = *sendS*2
→ *stateSNext t* = *waitS*
∧ *resp* (*Suc t*) = *Enc* (*kValue t*) (*Sign SKeyP* [*kE genKey*, *sE* (*uValue t*)]) ∧ *abortS* (*Suc t*) = [])))))”

constdefs
Server ::
“*InitMessage istream* ⇒ *Event istream* ⇒ *Expression istream* ⇒ *Expression istream* ⇒ *Event istream*
⇒ *bool*”
“*Server init abortC xchd resp abortS*
≡
∃ *st k u*.
Server_L init abortC xchd
(*fin_inf_append* [*initS*] *st*) (*fin_inf_append* [*SKey*] *k*) (*fin_inf_append* [*N*] *u*)
resp abortS st k u”

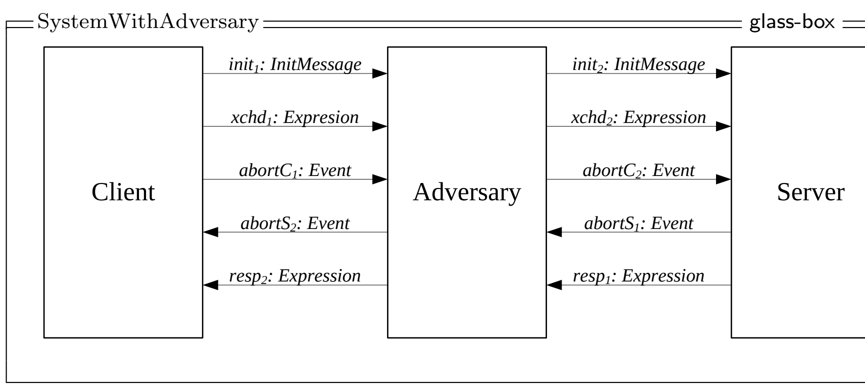
4.2 Security Analysis

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

Let $P = \mathit{Client} \otimes \mathit{Server}$. To show that P does not preserve the secrecy of $\mathit{secretD}$, $\mathit{secretD} \in \mathit{KS}$, we need to find an adversary component $\mathit{Adversary}$ with $I_{\mathit{Adversary}} \subseteq O_P$ such that $\mathsf{knows}^{\mathit{Adversary}}(m)$ holds with regards to the composition, and m does not belong to the set of private keys of $\mathit{Adversary}$ or to the set of unguessable values of $\mathit{Adversary}$:

$$\exists\, \mathit{Adversary} : I_{\mathit{Adversary}} \subseteq O_P \;\wedge\; m \notin \mathit{KS}_{\mathit{Adversary}} \;\wedge\; \mathsf{knows}^{\mathit{Adversary}}(m)$$

As mentioned in [SJ08], 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 $\mathit{Sign}(\mathit{CAKey}^{-1}, \langle S, \mathit{SKey}\rangle)$ signed by the certification authority that contains its name and public key. An adversary may also have access to CAKey, $\mathit{Sign}(\mathit{CAKey}^{-1}, \langle S, \mathit{SKey}\rangle)$ and $\mathit{Sign}(\mathit{CAKey}^{-1}, \langle Z, \mathit{ZKey}\rangle)$ for an arbitrary process Z.



Consider the Focus specification of the component $\mathit{Adversary}$ presented below. We used in this specification the following auxiliary data type: $\mathit{AdvStates} = \{\mathit{initA}, \mathit{sendA}1, \mathit{sendA}2\}$

Please note that this specification is weak causal: we assume that the adversary does not delay any message. Please also note that the presented here specification of the adversary component is different from one defined in [SJ08]: we corrected some mistakes and change the component specification to be more readable.

The value $\mathit{genKey} \in \mathit{Keys}$ is a session key, which is symmetric (i.e. $\mathit{genKey}^{-1} = \mathit{genKey}$) and is generated by the server. This implies that

$\mathsf{knows}^{\mathit{Adversary}}(\mathit{genKey})$

holds if and only if

$$\mathsf{knows}^{Adversary}(genKey^{-1})$$

holds. Thus, if the adversary knows the value of *genKey* it also knows the value of $genKey^{-1}$.

If we trace its knowledge base as its evolves in interaction with the protocol components, we get that *Adversary* will know the secret *secretD* at the time unit 4. Let us discuss the data flows more precisely, step by step, representing them by timed tables for the time intervals $[0, \dots, 4]$ (we represent only the values of the output streams and the local variables of the components).

First of all computations, let us make the auxiliary computations:

$$\begin{array}{l} ungValue(\mathsf{ft}.\mathsf{ti}(init_2, 0)) = \\ ungValue(im(N, AKey, Sign(AKey^{-1}, \langle C, AKey \rangle))) = \\ N \end{array}$$

$$\begin{array}{l} key(\mathsf{ft}.\mathsf{ti}(init_2, 0)) = \\ key(im(N, AKey, Sign(AKey^{-1}, \langle C, AKey \rangle))) = \\ AKey \end{array}$$

$$\begin{array}{l} \mathsf{ft}.Ext(CAKey, \mathsf{ti}(resp_2, 2)) = \\ \mathsf{ft}.Ext(CAKey, Sign(CAKey^{-1}, \langle S, SKey \rangle)) = \\ \mathsf{ft}.\langle S, SKey \rangle = \\ S \end{array}$$

$$\begin{array}{l} \mathsf{snd}.Ext(CAKey, \mathsf{ti}(resp_2, 2)) = \\ \mathsf{snd}.Ext(CAKey, Sign(CAKey^{-1}, \langle S, SKey \rangle)) = \\ SKey \end{array}$$

$$\begin{array}{l} Ext(SKey, Decr(CKey^{-1}, \mathsf{ti}(resp_2, 3))) = \\ Ext(SKey, Decr(CKey^{-1}, Enc(CKey, Sign(SKey^{-1}, \langle genKey, N \rangle)))) = \\ Ext(SKey, Sign(SKey^{-1}, \langle genKey, N \rangle)) = \\ \langle genKey, N \rangle \end{array}$$

Adversary — timed

in $abortC_1, abortS_1 : Event;\ xchd_1, resp_1 : Expression;\ init_1 : InitMessage$

out $abortC_2, abortS_2 : Event;\ xchd_2, resp_2 : Expression;\ init_2 : InitMessage$

local $aCKey, aSKey, aKey \in Keys;\ stateA \in AdvStates$

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

gar

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

1 $\mathsf{ti}(abortC_2, t) = \mathsf{ti}(abortC_1, t)$

2 $\mathsf{ti}(abortS_2, t) = \mathsf{ti}(abortS_1, t)$

3 $\mathsf{ti}(init_1, t) \neq \langle\rangle \rightarrow$
$aCKey' = key((init_1)^t_{\mathsf{ft}}) \wedge$
$\mathsf{ti}(init_2, t) = \langle im(ungValue((init_1)^t_{\mathsf{ft}}), AKey, Sign(AKey^{-1}, \langle C, AKey\rangle))\rangle$

4 $\mathsf{ti}(init_1, t) = \langle\rangle \rightarrow \mathsf{ti}(init_2, t) = \langle\rangle \wedge aCKey' = aCKey$

5 $\mathsf{ti}(resp_1, t) = \langle\rangle \rightarrow$
$stateA' = initA \wedge aSKey' = aSKey \wedge aKey' = aKey \wedge \mathsf{ti}(resp_2, t) = \langle\rangle$

6 $\mathsf{ti}(resp_1, t) \neq \langle\rangle \wedge stateA = initA \rightarrow$
$stateA' = sendA1 \wedge aSKey' = aSKey \wedge aKey' = aKey \wedge \mathsf{ti}(resp_2, t) = \mathsf{ti}(resp_1, t)$

7 $\mathsf{ti}(resp_1, t) \neq \langle\rangle \wedge stateA = sendA1 \rightarrow$
$stateA' = sendA2 \wedge aSKey' = \mathsf{snd}.Ext(CAKey, \mathsf{ti}(resp_1, t)) \wedge$
$aKey' = aKey \wedge$
$\mathsf{ti}(resp_2, t) = \mathsf{ti}(resp_1, t)$

8 $\mathsf{ti}(resp_1, t) \neq \langle\rangle \wedge stateA = sendA2 \rightarrow$
$stateA' = initA \wedge aSKey' = aSKey \wedge$
$aKey = \mathsf{ft}.Ext(aSKey, Decr(AKey^{-1}, \mathsf{ti}(resp_1, t)))$
$\mathsf{ti}(resp_2, t) = Enc(aCKey, Decr(AKey^{-1}, \mathsf{ti}(resp_1, t)))$

9 $\mathsf{ti}(xchd_2, t) = \mathsf{ti}(xchd_1, t)$

Translating the Focus specifications to Isabelle/HOL according to the methodology "Focus on Isabelle" we can prove formally that the security flaw exists. These proof (together with protocol component specifications and auxiliary lemmas) takes 1,5 kloc. In this report we present only the Isabelle/HOL specification of the components and the main lemma (without the proof) which says that the during the 4th time unit the secret data *secretD* will be send to the adversary by the *Client* component and no abort-signal will be produced. For further details we would like to refer to the Isabelle/HOL-theory *Handshake-Protocol.thy*.

Client:

t	$init_1$	$xchd_1$	$abortC_1$	$check$	enc
0	$\langle im(N, CKey, Sign(CKey^{-1}, \langle C, CKey\rangle))\rangle$	$\langle\rangle$	$\langle\rangle$	st0	
1	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st0	
2	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st1	
3	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st2	SKey
4	$\langle\rangle$	$Enc(genKey, secretD)$	$\langle\rangle$	st0	SKey

Server:

t	$resp_1$	$abortS_1$	$stateS$	$kValue$	$uValue$
0	$\langle\rangle$	$\langle\rangle$	$initS$		
1	$\langle N\rangle$	$\langle\rangle$	$sendS1$	$AKey$	N
2	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$	$\langle\rangle$	$sendS2$	$AKey$	N
3	$Enc(AKey, Sign(SKey^{-1}, \langle genKey, N\rangle))$	$\langle\rangle$	$waitS$	$AKey$	N
4	$\langle\rangle$	$\langle\rangle$	$waitS$	$AKey$	N

Adversary:

t	$resp_2$	$init_2$	$xchd_2$	$abortC_2$	$abortS_2$	knows^A
0	$\langle\rangle$	$\langle im(N, AKey, Sign(AKey^{-1}, \langle C, AKey\rangle))\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$CAKey,\ AKey,\ AKey^{-1}$ $N,\ CKey$
1	$\langle N\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	
2	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$ $SKey$
3	$Enc(CKey, Sign(SKey^{-1}, \langle genKey, N\rangle))$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$Sign(SKey^{-1}, \langle genKey, N\rangle)$ $genKey,\ genKey^{-1}$
4	$\langle\rangle$	$\langle\rangle$	$Enc(genKey, secretD)$	$\langle\rangle$	$\langle\rangle$	$Enc(genKey, secretD)$ $secretD$

The Isabelle/HOL specifications of the component *Client* and *Server* are presented in Section 4, in this section we show the Isabelle/HOL specification of the adversary component and the main lemma itself.

```
constdefs
Adv_L ::
"Event istream ⇒ Event istream ⇒
Expression istream ⇒ Expression istream ⇒  initMessage istream ⇒
Keys iustream ⇒ Keys iustream ⇒ Keys iustream ⇒ AdvStates iustream ⇒
Event istream ⇒ Event istream ⇒
Expression istream ⇒ Expression istream ⇒  initMessage istream ⇒
Keys iustream ⇒ Keys iustream ⇒ Keys iustream ⇒ AdvStates iustream ⇒ bool"

"Adv_L abortC1 abortS1 xchd1 resp1 init1  aCKey_in aSKey_in aKey_in stateA_in
         abortC2 abortS2 xchd2 resp2 init2  aCKey_out aSKey_out aKey_out stateA_out
≡
(((msg (1 :: nat) init1) ∧ (msg (1 :: nat) xchd1))
→
(∀(t :: nat).(
(abortC2 t = abortC1 t)
∧
(abortS2 t = abortS1 t)
∧
((init1 t ≠ [])
→ aCKey_out t = (key (hd (init1 t)))
∧ init2 t = [(| ungValue = ungValue (hd (init1 t)), key = AKey, imsg = (SignAKeyP[idEsClient, kEAKey]) |)])
∧
(init1 t = []
→ aCKey_out t = aCKey_in t ∧ init2 t = [])
∧
(resp1 t = []
→ stateA_out t = initA ∧ aSKey_out t = aSKey_in t
∧ aKey_out t = aKey_in t ∧ resp2 t = [])
∧
(resp1 t ≠ [] ∧ stateA_in t = initA
→ stateA_out t = sendA1 ∧ aSKey_out t = aSKey_in t
∧ aKey_out t = aKey_in t ∧ resp2 t = resp1 t)
∧
(resp1 t ≠ [] ∧ stateA_in t = sendA1
→ stateA_out t = sendA2 ∧ aSKey_out t = Expr2Keys(hd(tl(ExtCAKey(resp1 t))))
∧ aKey_out t = aKey_in t ∧ resp2 t = resp1 t)
∧
(resp1 t ≠ [] ∧ stateA_in t = sendA2
→ stateA_out t = initA ∧ aSKey_out t = aSKey_in t
∧ aKey_out t =  Expr2Keys (hd (Ext (aSKey_in t) (Decr AKeyP (resp1 t))))
∧ resp2 t = Enc (aCKey_in t) (Decr AKeyP (resp1 t)))
∧
(xchd2 t = xchd1 t))))"
```

constdefs
Adv ::
"*Event istream* ⇒ *Event istream* ⇒
Expression istream ⇒ *Expression istream* ⇒ *initMessage istream* ⇒
Event istream ⇒ *Event istream* ⇒
Expression istream ⇒ *Expression istream* ⇒ *initMessage istream* ⇒ *bool*"

"*Adv abortC*1 *abortS*1 *xchd*1 *resp*1 *init*1 *abortC*2 *abortS*2 *xchd*2 *resp*2 *init*2
≡
∃ *aCKey aSKey aKey stateA*.
*Adv_L abortC*1 *abortS*1 *xchd*1 *resp*1 *init*1
(*fin_inf_append* [*AKey*] *aCKey*)
(*fin_inf_append* [*AKey*] *aSKey*)
(*fin_inf_append* [*AKey*] *aKey*)
(*fin_inf_append* [*initA*] *stateA*)
*abortC*2 *abortS*2 *xchd*2 *resp*2 *init*2
aCKey aSKey aKey stateA"

lemma *test_security_flaw* :
"[| *Client abortS*2 *resp*2 *init*1 *xchd*1 *abortC*1;
*Server init*2 *abortC*2 *xchd*2 *resp*1 *abortS*1;
*Adv abortC*1 *abortS*1 *xchd*1 *resp*1 *init*1 *abortC*2 *abortS*2 *xchd*2 *resp*2 *init*2 |]
⇒
*abortC*1 (4 :: *nat*) = []
∧
*xchd*1 (4 :: *nat*) = *Enc genKey* [*sE secretD*]"

4.3 Fixing the Security Weakness

To fix this security weakness (vs. both kinds of adversary), 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:

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

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

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

We mark these changes yellow in the Focus specifications.

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.

We omit in this report the presentation of the Isabelle/HOL specification of the corrected components *Client* and *Server*, because they have only the minor changes vs. the specification presented in Section 4 and these changes are clearly shown in the corresponding Focus specifications. Thus, we discuss here only the main lemma (without the proof) which says that the during the 4th time unit no secret data *secretD* will be send to the adversary by the *Client* component and the abort-signal will be produced at this time unit. The proof (together with protocol component specifications and auxiliary lemmas) takes also about 1,5 kloc, but the most number of lemmas is the same as for the uncorrected version referred above in theory *HandshakeProtocol.thy*. For further details on the corrected version of the protocol we would like to refer to the Isabelle/HOL-theory *HandshakeProtocolCorrected.thy*.

```
lemma test_security_flaw :
"[| Client abortS2 resp2 init1 xchd1 abortC1;
  Server init2 abortC2 xchd2 resp1 abortS1;
  Adv abortC1 abortS1 xchd1 resp1 init1 abortC2 abortS2 xchd2 resp2 init2 |]
  ⇒
 abortC1 (4 :: nat) = [event]
 ∧
 xchd1 (4 :: nat) = []"
```

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*.

Client — timed

in $abortS : Event;\ resp : Expression$

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

local $check : StateC;\ enc : Keys$

init $check = st0;$

asm true

gar

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

2 $\mathsf{ti}(xchd, 0) = \langle\rangle$

3 $\mathsf{ti}(abortC, 0) = \langle\rangle$

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

4 $\mathsf{ti}(init, t+1) = \langle\rangle$

5 $\mathsf{ti}(abortS, t) \neq \langle\rangle \rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t+1) = \langle\rangle \wedge check' = st0$

6 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) = \langle\rangle \wedge check = st0$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = st0$

7 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check' = st0$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = st1$

8 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check = st1 \wedge \mathsf{ft}.secr = S$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = st2 \wedge enc' = \mathsf{snd}.secr$

9 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check = st2 \wedge \mathsf{snd}.res = N \wedge \mathsf{trd}.res = CKey$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t+1) = Enc(\mathsf{ft}.res, secretD) \wedge check' = st0$

10 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge$
$((check = st1 \wedge (\mathsf{ti}(resp, t) = \langle\rangle \vee (\mathsf{ti}(resp, t) \neq \langle\rangle \wedge \mathsf{ft}.secr \neq S))) \vee$
$(check = st2 \wedge (\mathsf{ti}(resp, t) = \langle\rangle \vee (\mathsf{ti}(resp, t) \neq \langle\rangle \wedge \mathsf{snd}.res \neq N \vee \mathsf{trd}.res = CKey))))$
$\rightarrow \mathsf{ti}(abortC, t+1) = \langle event\rangle \wedge$
$\mathsf{ti}(xchd, t+1) = \langle\rangle \wedge$
$check' = st0$

where
$secr = Ext(CAKey, \mathsf{ti}(resp, t))$
$res = Ext(enc, Decr(CKey^{-1}, \mathsf{ti}(resp, t)))$

Server (timed)

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

out $resp : Expression;\ abortS : Event$

---

local $stateS \in StateS;\ kValue \in Keys;\ uValue \in Secret$

init $stateS = initS$

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

gar

1 $\mathsf{ti}(resp, 0) = \langle\rangle$

2 $\mathsf{ti}(abortS, 0) = \langle\rangle$

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

3 $\mathsf{ti}(abortC, t) \neq \langle\rangle$
$\rightarrow stateS' = initS \ \wedge\ \mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

4 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = waitS$
$\rightarrow \mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle \ \wedge stateS' = waitS$

5 $\mathsf{ti}(abortC, t) = \langle\rangle \wedge stateS = initS \wedge \mathsf{ti}(init, t) = \langle\rangle$
$\rightarrow \mathsf{ti}(resp, t+1) = \langle\rangle \wedge stateS' = initS$

6 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = initS \ \wedge\ \mathsf{ti}(init, t) \neq \langle\rangle \ \wedge$
$\mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) \neq key(init^t_{\mathsf{ft}})$
$\rightarrow \mathsf{ti}(resp, t+1) = \langle\rangle \ \wedge\ stateS' = initS \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle event\rangle$

7 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = initS \ \wedge\ \mathsf{ti}(init, t) \neq \langle\rangle \ \wedge$
$\mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) = key(init^t_{\mathsf{ft}})$
$\rightarrow \mathsf{ti}(resp, t+1) = \langle ungValue(init^t_{\mathsf{ft}})\rangle$
$\wedge\ stateS' = sendS1 \ \wedge\ uValue' = ungValue(init^t_{\mathsf{ft}}) \ \wedge\ kValue' = key(init^t_{\mathsf{ft}})$
$\wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

8 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = sendS1$
$\rightarrow \mathsf{ti}(resp, t+1) = Sign(CAKey^{-1}, \langle S, SKey\rangle)$
$\wedge\ stateS' = sendS2 \ \wedge\ uValue' = uValue \ \wedge\ kValue' = kValue$
$\wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

9 $\mathsf{ti}(abortC, t) = \langle\rangle \ \wedge\ stateS = sendS2$
$\rightarrow \mathsf{ti}(resp, t+1) = Enc(kValue, Sign(SKey^{-1}, \langle genKey,\ kValue,\ uValue\rangle))$
$\wedge\ stateS' = waitS \ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle$

Now, if we trace the knowledge base of the adversary *Adversary* considered above, the secret is not leaked, the transmission will be aborted by the client: see the communication history for the time intervals $[0, \ldots, 4]$.

Client:

t	$init_1$	$xchd_1$	$abortC_1$	$check$	enc
0	$\langle im(N, CKey, Sign(CKey^{-1}, \langle C, CKey\rangle))\rangle$	$\langle\rangle$	$\langle\rangle$	st0	
1	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st0	
2	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st1	
3	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st2	SKey
4	$\langle\rangle$	$\langle\rangle$	$\langle event\rangle$	st0	SKey

Server:

t	$resp_1$	$abortS_1$	$stateS$	$kValue$	$uValue$
0	$\langle\rangle$	$\langle\rangle$	$initS$		
1	$\langle N\rangle$	$\langle\rangle$	$sendS1$	$AKey$	N
2	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$	$\langle\rangle$	$sendS2$	$AKey$	N
3	$Enc(AKey, Sign(SKey^{-1}, \langle genKey, AKey, N\rangle))$	$\langle\rangle$	$waitS$	$AKey$	N
4	$\langle\rangle$	$\langle\rangle$	$waitS$	$AKey$	N

Adversary:

t	$resp_2$	$init_2$	$xchd_2$	$abortC_2$	$abortS_2$	knows^A
0	$\langle\rangle$	$\langle im(N, AKey, Sign(AKey^{-1}, \langle C, AKey\rangle))\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$CAKey,\ AKey,\ AKey^{-1}$ $N,\ CKey$
1	$\langle N\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	
2	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$ $SKey$
3	$Enc(CKey, Sign(SKey^{-1}, \langle genKey, AKey, N\rangle))$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$Sign(SKey^{-1}, \langle genKey, AKey, N\rangle)$ $genKey,\ genKey^{-1}$
4	$\langle\rangle$	$\langle\rangle$	$\langle event\rangle$	$\langle\rangle$	$\langle\rangle$	

4.4 Open Question

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. If these properties does not hold, i.e. if an adversary can obtain the key $CAKey^{-1}$, the protocol cannot guarantee the security properties anymore. For this case is also not really important whether the value $genKey \in Keys$ is symmetric or not, an adversary can dead also with an asymmetric key (i.e. with the keys $genKey^{-1} \neq genKey$).

If the adversary knows $CAKey^{-1}$, it can replace the server key $SKey$ on the time interval 2 by its own key $AKey$: the value of $\mathsf{ti}(resp_2, 2)$ will be now not

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

but

$$Sign(CAKey^{-1}, \langle S, AKey \rangle).$$

At the next step the adversary will output also a different (vs. the first version of the adversary specification) message. The adversary will use $AKey^{-1}$ and $genKeyA$ (own session key) to generate $Sign(AKey^{-1}, \langle genKeyA, N \rangle)$ instead to resend the $Sign(SKey^{-1}, \langle genKey, N \rangle)$ to the client component. Thus, the value of $\mathsf{ti}(resp_2, 3)$ will be now not

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

but

$$Enc(CKey, Sign(AKey^{-1}, \langle genKeyA, N \rangle)).$$

This implies also that the client component will save the $AKey$ as the server key.

If we trace its knowledge base as its evolves in interaction with the protocol components, we get that also this version of an *Adversary* component will know the secret *secretD* at the time unit 4 also if we use the extended versions of the client and server components.

Adversary — timed

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

local $aCKey, aSKey, aKey \in Keys;\ aN, aSecret \in Secret;$
$stateA \in AdvStates$

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

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

1 $\mathsf{ti}(abortC_2, t) = \mathsf{ti}(abortC_1, t)$

2 $\mathsf{ti}(abortS_2, t) = \mathsf{ti}(abortS_1, t)$

3 $\mathsf{ti}(init_1, t) \neq \langle\rangle \ \rightarrow$
$aCKey' = key((init_1)^t_{\mathsf{ft}}) \ \wedge$
$\mathsf{ti}(init_2, t) = \langle im(ungValue((init_1)^t_{\mathsf{ft}}), AKey, Sign(AKey^{-1}, \langle C, AKey\rangle))\rangle$

4 $\mathsf{ti}(init_1, t) = \langle\rangle \ \rightarrow\ \mathsf{ti}(init_2, t) = \langle\rangle \ \wedge\ aCKey' = aCKey$

5 $\mathsf{ti}(resp_1, t) = \langle\rangle \ \rightarrow$
$stateA' = initA \ \wedge\ aSKey' = aSKey \ \wedge\ aKey' = aKey \ \wedge\ \mathsf{ti}(resp_2, t) = \langle\rangle$

6 $\mathsf{ti}(resp_1, t) \neq \langle\rangle \wedge stateA = initA \ \rightarrow$
$stateA' = sendA1 \ \wedge\ aSKey' = aSKey \ \wedge\ aKey' = aKey \ \wedge\ \mathsf{ti}(resp_2, t) = \mathsf{ti}(resp_1, t)$

7 $\mathsf{ti}(resp_1, t) \neq \langle\rangle \wedge stateA = sendA1 \ \rightarrow$
$stateA' = sendA2 \ \wedge\ aSKey' = \mathsf{snd}.Ext(CAKey, \mathsf{ti}(resp_1, t)) \ \wedge$
$aKey' = aKey \ \wedge$
$\mathsf{ti}(resp_2, t) = Sign(CAKey^{-1}, \langle \mathsf{ft}.Ext(CAKey, \mathsf{ti}(resp_1, t)), AKey\rangle)$

8 $\mathsf{ti}(resp_1, t) \neq \langle\rangle \wedge stateA = sendA2 \ \rightarrow$
$stateA' = initA \ \wedge\ aSKey' = aSKey \ \wedge$
$aKey = \mathsf{ft}.Ext(aSKey, Decr(AKey^{-1}, \mathsf{ti}(resp_1, t)))$
$\mathsf{ti}(resp_2, t) = Enc(aCKey, Sign(AKey^{-1}, \langle genKeyA, aCKey, aN\rangle))$

9 $\mathsf{ti}(xchd_1, t) = \langle\rangle \ \rightarrow aSecret' = aSecret \wedge \mathsf{ti}(xchd_2, t) = \langle\rangle$

10 $\mathsf{ti}(xchd_1, t) \neq \langle\rangle \ \rightarrow$
$aSecret' = Decr(genKeyA^{-1}, \mathsf{ti}(xchd_1, t)) \wedge \mathsf{ti}(xchd_2, t) = Enc(aKey, aSecret')$

Client:

t	$init_1$	$xchd_1$	$abortC_1$	$check$	enc
0	$\langle im(N, CKey, Sign(CKey^{-1}, \langle C, CKey\rangle))\rangle$	$\langle\rangle$	$\langle\rangle$	st0	
1	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st0	
2	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st1	
3	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	st2	AKey
4	$\langle\rangle$	$Enc(genKeyA, secretD)$	$\langle\rangle$	st0	AKey

Server:

t	$resp_1$	$abortS_1$	$stateS$	$kValue$	$uValue$
0	$\langle\rangle$	$\langle\rangle$	$initS$		
1	$\langle N\rangle$	$\langle\rangle$	$sendS1$	$AKey$	N
2	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$	$\langle\rangle$	$sendS2$	$AKey$	N
3	$Enc(AKey, Sign(SKey^{-1}, \langle genKey, AKey, N\rangle))$	$\langle\rangle$	$waitS$	$AKey$	N
4	$\langle\rangle$	$\langle\rangle$	$waitS$	$AKey$	N

Adversary:

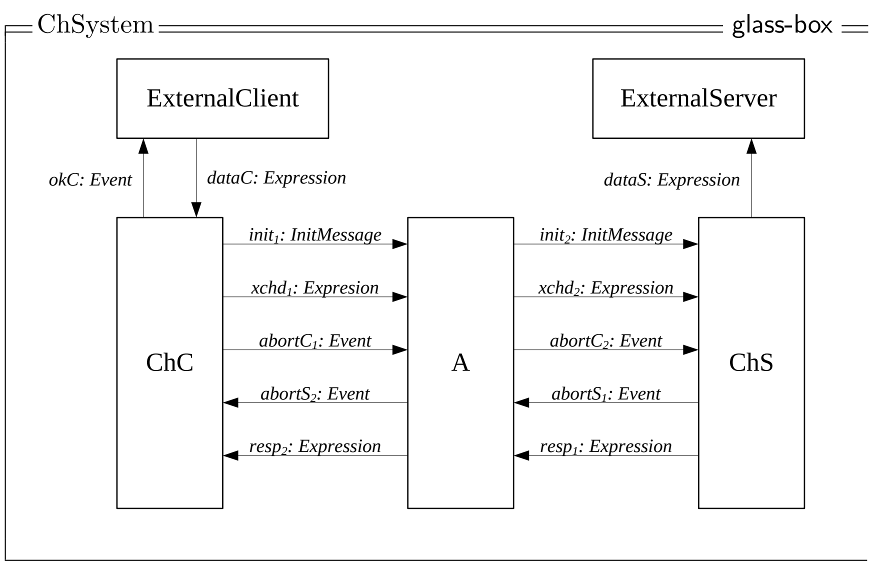
t	$resp_2$	$init_2$	$xchd_2$	$abortC_2$	$abortS_2$	knows^A
0	$\langle\rangle$	$\langle im(N, AKey, Sign(AKey^{-1}, \langle C, AKey\rangle))\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$CAKey,\ CAKey^{-1},$ $AKey,\ AKey^{-1}$ $genKeyA,\ genKeyA^{-1},\ N,\ CKey$
1	$\langle N\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	
2	$Sign(CAKey^{-1}, \langle S, AKey\rangle)$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$ $SKey$
3	$Enc(CKey, Sign(AKey^{-1},$ $\langle genKeyA, CKey, N\rangle))$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$Sign(SKey^{-1}, \langle genKey, AKey, N\rangle)$ $genKey$
4	$\langle\rangle$	$\langle\rangle$	$Enc(genKey, secretD)$	$\langle\rangle$	$\langle\rangle$	$Enc(genKeyA, secretD),\ secretD$

5 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 4.2). 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 $t+2+delay$, where *delay* is a communication delay dependent on the communication medium, and the two time units delay arises from using the secure channels. We specify here a system with optimized (vs. the version from [SJ08]) secure channel components in Focus as a composed component *ChSystem* and additionally present for both components, *ChC* and *ChS*, the corresponding timed table.



ChC — timed

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

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

local $\quad check \in \{0,1,2,3\};\quad buffer \in Expression^*;\ enc \in Keys$

init $\quad check = 0;\quad buffer = \langle\rangle$

asm $\quad \mathsf{msg}_1(dataC)$

gar

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

2 $\mathsf{ti}(xchd, 0) = \langle\rangle$

3 $\mathsf{ti}(abortC, 0) = \langle\rangle$

4 $\mathsf{ti}(okC, 0) = \langle\rangle$

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

5 $\mathsf{ti}(init, t+1) = \langle\rangle$

6 $\mathsf{ti}(abortS, t) \neq \langle\rangle$
$\to \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t+1) = \langle\rangle \wedge check' = 0 \wedge \mathsf{ti}(okC, t+1) = \langle\rangle$

7 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) = \langle\rangle \wedge check = 0$
$\to \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = 0 \wedge \mathsf{ti}(okC, t+1) = \langle\rangle$

8 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check = 0$
$\to \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = 1 \wedge \mathsf{ti}(okC, t+1) = \langle\rangle$

9 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check = 1 \wedge \mathsf{ft}.secr = S$
$\to \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge \mathsf{ti}(xchd, t) = \langle\rangle \wedge check' = 2 \wedge enc' = \mathsf{snd}.secr \wedge \mathsf{ti}(okC, t+1) = \langle\rangle$

10 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge \mathsf{ti}(resp, t) \neq \langle\rangle \wedge check = 2 \wedge \mathsf{snd}.res = N \wedge \mathsf{trd}.res = CKey$
$\to \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge$
$\mathsf{ti}(xchd, t+1) = Enc(\mathsf{ft}.res, secretD) \wedge$
$check' = 3 \wedge \mathsf{ti}(okC, t+1) = \langle\rangle \wedge enc' = enc$

11 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge$
$((check = 1 \wedge (\mathsf{ti}(resp, t) = \langle\rangle \vee (\mathsf{ti}(resp, t) \neq \langle\rangle \wedge \mathsf{ft}.secr \neq S))) \vee$
$(check = 2 \wedge (\mathsf{ti}(resp, t) = \langle\rangle \vee (\mathsf{ti}(resp, t) \neq \langle\rangle \wedge (\mathsf{snd}.res \neq N \vee \mathsf{trd}.res = CKey)))))$
$\to \mathsf{ti}(abortC, t+1) = \langle event\rangle \wedge$
$\mathsf{ti}(xchd, t+1) = \langle\rangle \wedge check' = 0 \wedge \mathsf{ti}(okC, t+1) = \langle\rangle$

12 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge check = 3 \wedge buffer = \langle\rangle \wedge \mathsf{ti}(dataC, t) = \langle\rangle$
$\to \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge$
$\mathsf{ti}(xchd, t+1) = \langle\rangle \wedge check' = 3 \wedge \mathsf{ti}(okC, t+1) = \langle\rangle \wedge buffer' = \langle\rangle$

13 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge check = 3 \wedge buffer = \langle\rangle \wedge \mathsf{ti}(dataC, t) \neq \langle\rangle$
$\to \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge$
$\mathsf{ti}(xchd, t+1) = Enc(enc, \mathsf{ti}(dataC, t)) \wedge$
$check' = 3 \wedge \mathsf{ti}(okC, t+1) = \langle event\rangle \wedge buffer' = \langle\rangle$

14 $\mathsf{ti}(abortS, t) = \langle\rangle \wedge check = 3 \wedge buffer \neq \langle\rangle$
$\to \mathsf{ti}(abortC, t+1) = \langle\rangle \wedge$
$\mathsf{ti}(xchd, t+1) = Enc(enc, \mathsf{ft}.buffer) \wedge$
$check' = 3 \wedge \mathsf{ti}(okC, t+1) = \langle event\rangle \wedge buffer' = \mathsf{rt}.buffer \frown \mathsf{ti}(dataC, t)$

15 $check = 3 \to enc' = enc$

16 $check \neq 3 \to buffer' = buffer \frown \mathsf{ti}(dataC, t)$

where
$secr = Ext(CAKey, \mathsf{ti}(resp, t))$
$res = Ext(enc, Decr(CKey^{-1}, \mathsf{ti}(resp, t)))$

ChS — timed

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

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

---

local $\quad stateS \in StateS;\ kValue \in Keys;\ uValue \in Secret$

init $\quad stateS = initS$

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

---

gar

1 $\quad \mathsf{ti}(resp, 0) = \langle\rangle$

2 $\quad \mathsf{ti}(abortS, 0) = \langle\rangle$

3 $\quad \mathsf{ti}(dataS, 0) = \langle\rangle$

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

4 $\quad \mathsf{ti}(abortC, t) \neq \langle\rangle$
$\to stateS' = initS\ \wedge\ \mathsf{ti}(resp, t+1) = \langle\rangle\ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle\ \wedge$
$\mathsf{ti}(dataS, t+1) = \langle\rangle$

5 $\quad \mathsf{ti}(abortC, t) = \langle\rangle\ \wedge\ stateS = waitS\ \wedge\ \mathsf{ti}(xchd, t) = \langle\rangle$
$\to stateS' = waitS \wedge \mathsf{ti}(resp, t+1) = \langle\rangle\ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle\ \wedge$
$\mathsf{ti}(dataS, t+1) = \langle\rangle$

6 $\quad \mathsf{ti}(abortC, t) = \langle\rangle\ \wedge\ stateS = waitS\ \wedge\ \mathsf{ti}(xchd, t) \neq \langle\rangle$
$\to stateS' = waitS \wedge \mathsf{ti}(resp, t+1) = \langle\rangle\ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle\ \wedge$
$\mathsf{ti}(dataS, t+1) = Decr(genKey^{-1}, \mathsf{ti}(xchd, t))$

7 $\quad \mathsf{ti}(abortC, t) = \langle\rangle \wedge stateS = initS \wedge \mathsf{ti}(init, t) = \langle\rangle$
$\to stateS' = initS \wedge \mathsf{ti}(resp, t+1) = \langle\rangle \wedge \mathsf{ti}(abortS, t+1) = \langle\rangle\ \wedge$
$\mathsf{ti}(dataS, t+1) = \langle\rangle$

8 $\quad \mathsf{ti}(abortC, t) = \langle\rangle\ \wedge\ stateS = initS\ \wedge\ \mathsf{ti}(init, t) \neq \langle\rangle\ \wedge$
$\mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) \neq key(init^t_{\mathsf{ft}})$
$\to \mathsf{ti}(resp, t+1) = \langle\rangle\ \wedge\ stateS' = initS\ \wedge\ \mathsf{ti}(abortS, t+1) = \langle event\rangle\ \wedge$
$\mathsf{ti}(dataS, t+1) = \langle\rangle$

9 $\quad \mathsf{ti}(abortC, t) = \langle\rangle\ \wedge\ stateS = initS\ \wedge\ \mathsf{ti}(init, t) \neq \langle\rangle\ \wedge$
$\mathsf{snd}.Ext(\langle key(init^t_{\mathsf{ft}}), msg(init^t_{\mathsf{ft}})\rangle) = key(init^t_{\mathsf{ft}})$
$\to \mathsf{ti}(resp, t+1) = \langle ungValue(init^t_{\mathsf{ft}})\rangle$
$\wedge\ stateS' = sendS1\ \wedge\ uValue' = ungValue(init^t_{\mathsf{ft}})\ \wedge\ kValue' = key(init^t_{\mathsf{ft}})$
$\wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle \wedge \mathsf{ti}(dataS, t+1) = \langle\rangle$

10 $\quad \mathsf{ti}(abortC, t) = \langle\rangle\ \wedge\ stateS = sendS1$
$\to \mathsf{ti}(resp, t+1) = Sign(CAKey^{-1}, \langle S, SKey\rangle)$
$\wedge\ stateS' = sendS2\ \wedge\ uValue' = uValue\ \wedge\ kValue' = kValue$
$\wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle \wedge \mathsf{ti}(dataS, t+1) = \langle\rangle$

11 $\quad \mathsf{ti}(abortC, t) = \langle\rangle\ \wedge\ stateS = sendS2$
$\to \mathsf{ti}(resp, t+1) = Enc(kValue, Sign(SKey^{-1}, \langle genKey, kValue, uValue\rangle))$
$\wedge\ stateS' = waitS\ \wedge\ \mathsf{ti}(abortS, t+1) = \langle\rangle \wedge \mathsf{ti}(dataS, t+1) = \langle\rangle$

ChC — timed

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

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

local $check \in \{0,1,2,3\};\ \ buffer \in Expression^*;\ \ enc \in Keys$

init $check = 0;\ \ buffer = \langle\rangle$

asm $\mathsf{msg}_1(dataC)$

gar

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

2 $\mathsf{ti}(xchd, 0) = \langle\rangle$

3 $\mathsf{ti}(abortC, 0) = \langle\rangle$

4 $\mathsf{ti}(okC, 0) = \langle\rangle$

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

5 $\mathsf{ti}(init, t+1) = \langle\rangle$

tiTable *ChClientTable*

ChS — timed

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

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

local $stateS \in StateS;\ kValue \in Keys;\ uValue \in Secret$

init $stateS = initS$

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

gar

1 $\mathsf{ti}(resp, 0) = \langle\rangle$

2 $\mathsf{ti}(abortS, 0) = \langle\rangle$

3 $\mathsf{ti}(dataS, 0) = \langle\rangle$

tiTable *ChServerTable*

tiTable ChClientTable (univ $a : Event^*; \quad r, x : Expression^*$): $\forall\, t \in \mathbb{N}$

	$abortS$	$resp$	$dataC$	$xchd'$	$abortC'$	okC'	$check'$	enc'	$buffer'$	Assumption
1	a		x	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	0	enc	$buffer \frown x$	$a \neq \langle\rangle$
2	$\langle\rangle$	$\langle\rangle$	x	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	0	enc	$buffer \frown x$	$check = 0$
3	$\langle\rangle$	r	x	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	1	enc	$buffer \frown x$	$check = 0,\ r \neq \langle\rangle$
4	$\langle\rangle$	r	x	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	2	$\mathsf{snd}.secr$	$buffer \frown x$	$check = 1,\ r \neq \langle\rangle,\ \mathsf{ft}.secr = S$
5	$\langle\rangle$	r	x	$\langle\rangle$	$\langle event\rangle$	$\langle\rangle$	0	enc	$buffer \frown x$	$check = 1,\ r = \langle\rangle \vee \mathsf{ft}.secr = S$
6	$\langle\rangle$	r	x	$Enc(\mathsf{ft}.res, secretD)$	$\langle\rangle$	$\langle\rangle$	3	enc	$buffer \frown x$	$check = 2,\ r \neq \langle\rangle$ $\mathsf{snd}.res = N,\ \mathsf{trd}.res = CKey$
7	$\langle\rangle$	r	x	$\langle\rangle$	$\langle event\rangle$	$\langle\rangle$	0	enc	$buffer \frown x$	$check = 2,$ $r = \langle\rangle \vee \mathsf{snd}.res \neq N \vee \mathsf{trd}.res = CKey$
8	$\langle\rangle$		$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	3	enc	$\langle\rangle$	$check = 3,\ buffer = \langle\rangle$
9	$\langle\rangle$		x	$Enc(enc, x)$	$\langle\rangle$	$\langle event\rangle$	3	enc	$\langle\rangle$	$check = 3,\ buffer = \langle\rangle,\ x \neq \langle\rangle$
10	$\langle\rangle$		x	$Enc(enc, \mathsf{ft}.buffer)$	$\langle\rangle$	$\langle event\rangle$	3	enc	$\mathsf{rt}.buffer \frown x$	$check = 3,\ buffer \neq \langle\rangle$

where
$secr = Ext(CAKey, \mathsf{ti}(resp, t))$
$res = Ext(enc, Decr(CKey^{-1}, \mathsf{ti}(resp, t)))$

tiTable ChServerTable (univ $a : Event^*;\ \ x : Expression^*;\ \ i : InitMessage$): $\forall\, t \in \mathbb{N}$

	init	*abortC*	*xchd*	$dataS'$	$resp'$	$abortS'$	$stateS'$	$kValue'$	$uValue'$	Assumption
1		a		$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	*initS*			$a \neq \langle\rangle$
2	*init*	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	*waitS*			$stateS = waitS$
3	*init*	$\langle\rangle$	x	$Decr(genKey^{-1}, x)$	$\langle\rangle$	$\langle\rangle$	*waitS*			$stateS = waitS,\ x \neq \langle\rangle$
4	$\langle\rangle$	$\langle\rangle$		$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	*initS*			$stateS = initS$
5	$\langle i\rangle$	$\langle\rangle$		$\langle\rangle$	$\langle\rangle$	$\langle event\rangle$	*initS*			$stateS = initS$ snd.$Ext(\langle key(i), msg(i)\rangle) \neq key(i)$
6	$\langle i\rangle$	$\langle\rangle$		$\langle\rangle$	$\langle ungValue(i)\rangle$	$\langle\rangle$	$sendS1$	$key(i)$	$ungValue(i)$	$stateS = initS$ snd.$Ext(\langle key(i), msg(i)\rangle) = key(i)$
7	*init*	$\langle\rangle$	*xchd*	$\langle\rangle$	$Sign(CAKey^{-1}, \langle S, SKey\rangle)$	$\langle\rangle$	$sendS2$	*kValue*	*uValue*	$stateS = sendS1$
8	*init*	$\langle\rangle$	*xchd*	$\langle\rangle$	$Enc(kValue,$ $Sign(SKey^{-1},$ $\langle genKey, kValue, uValue\rangle))$	$\langle\rangle$	*waitS*	*kValue*	*uValue*	$stateS = sendS2$

6 Conclusions

We presented in this report an optimized an refined methodology to specify 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 using the theorem prover Isabelle/HOL and the the approach "Focus on Isabelle" .

As a running example, a variant of the Internet security protocol TLS is presented. We analyzed the version of the protocol published in [SJ08], refined the Focus specification to be more readable and demonstrated a security flaw in this version using the extended approach and table representation as well has proved the existence of the flaw formally, using Isabelle/HOL.

We also used the extended approach to harden the protocol in a formal way, and showed how to construct a new version of the secure channel on the basis of the corrected formal specification of the protocol. The formal proof that the discussed flaw no more exist in this corrected version of the protocol was done also in Isabelle/HOL.

On the base of these protocol we specified secure channels that adopt the main protocol properties.

References

[APS99] 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.

[Bro97] M. Broy. Compositional refinement of interactive systems. *J. ACM*, 44(6):850–891, 1997.

[Bro98] 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.

[Bro05] Manfred Broy. Service-oriented systems engineering: Specification and design of services and layered architectures. The JANUS Approach. pages 47–81, July 2005.

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* Springer, 2001.

[DY83] 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.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[SJ08] M. Spichkova and J. Jürjens. Formal Specification of Cryptographic Protocols and Their Composition Properties: FOCUS-oriented approach. Technical report, Technische Universität München, 2008.

[Spi07] M. Spichkova. *Specification and Seamless Verification of Embedded Real-Time Systems: FOCUS on Isabelle*. PhD thesis, Technische Universität München, 2007.

[Spi11a] M. Spichkova. Focus on processes. Technical Report TUM-I1115, Technische Universität München, 2011.

[Spi11b] M. Spichkova. User Guide for the FOCUS representation in LaTeX. Technical Report TUM-I1119, Technische Universität München, 2011.