Compositional properties of crypto-based components

Maria Spichkova

February 6, 2026

Abstract

This paper presents an Isabelle/HOL [1] set of theories which allows to specify crypto-based components and to verify their composition properties wrt. cryptographic aspects. We introduce a formalisation of the security property of data secrecy, the corresponding definitions and proofs. A part of these definitions is based on [3].
Please note that here we import the Isabelle/HOL theory ListExtras.thy, presented in [2].

Contents

1 Auxiliary data types **2**

2 Correctness of the relations between sets of Input/Output channels **2**

3 Secrecy: Definitions and properties **4**

4 Local Secrets of a component **19**

5 Knowledge of Keys and Secrets **26**

1 Auxiliary data types

theory *Secrecy-types*
imports *Main*
begin

— We assume disjoint sets: Data of data values,
— Secrets of unguessable values, Keys - set of cryptographic keys.
— Based on these sets, we specify the sets EncType of encryptors that may be
— used for encryption or decryption, and Expression of expression items.
— The specification (component) identifiers should be listed in the set specID,
— the channel indentifiers should be listed in the set chanID.

datatype *Keys* = *CKey* | *CKeyP* | *SKey* | *SKeyP* | *genKey*
datatype *Secrets* = *secretD* | *N* | *NA*
type-synonym *Var* = *nat*
type-synonym *Data* = *nat*
datatype *KS* = *kKS Keys* | *sKS Secrets*
datatype *EncType* = *kEnc Keys* | *vEnc Var*
datatype *specID* = *sComp1* | *sComp2* | *sComp3* | *sComp4*
datatype *Expression* = *kE Keys* | *sE Secrets* | *dE Data* | *idE specID*
datatype *chanID* = *ch1* | *ch2* | *ch3* | *ch4*

primrec *Expression2KSL*:: *Expression list* ⇒ *KS list*
where
Expression2KSL [] = [] |
Expression2KSL (*x*#*xs*) =
((*case x of* (*kE m*) ⇒ [*kKS m*]
| (*sE m*) ⇒ [*sKS m*]
| (*dE m*) ⇒ []
| (*idE m*) ⇒ []) @ *Expression2KSL xs*)

primrec *KS2Expression*:: *KS* ⇒ *Expression*
where
KS2Expression (*kKS m*) = (*kE m*) |
KS2Expression (*sKS m*) = (*sE m*)

end

2 Correctness of the relations between sets of Input/Output channels

theory *inout*
imports *Secrecy-types*
begin

consts
subcomponents :: *specID* ⇒ *specID set*

— Mappings, defining sets of input, local, and output channels
— of a component
consts
ins :: *specID* ⇒ *chanID set*
loc :: *specID* ⇒ *chanID set*
out :: *specID* ⇒ *chanID set*

— Predicate insuring the correct mapping from the component identifier
— to the set of input channels of a component
definition
inStream :: *specID* ⇒ *chanID set* ⇒ *bool*
where
inStream x y ≡ (*ins x* = *y*)

— Predicate insuring the correct mapping from the component identifier
— to the set of local channels of a component
definition
locStream :: *specID* ⇒ *chanID set* ⇒ *bool*
where
locStream x y ≡ (*loc x* = *y*)

— Predicate insuring the correct mapping from the component identifier
— to the set of output channels of a component
definition
outStream :: *specID* ⇒ *chanID set* ⇒ *bool*
where
outStream x y ≡ (*out x* = *y*)

— Predicate insuring the correct relations between
— to the set of input, output and local channels of a component
definition
correctInOutLoc :: *specID* ⇒ *bool*
where
correctInOutLoc x ≡
(*ins x*) ∩ (*out x*) = {}
∧ (*ins x*) ∩ (*loc x*) = {}
∧ (*loc x*) ∩ (*out x*) = {}

— Predicate insuring the correct relations between
— sets of input channels within a composed component
definition
correctCompositionIn :: *specID* ⇒ *bool*
where
correctCompositionIn x ≡
(*ins x*) = (⋃ (*ins* ‘ (*subcomponents x*)) − (*loc x*))
∧ (*ins x*) ∩ (⋃ (*out* ‘ (*subcomponents x*))) = {}

— Predicate insuring the correct relations between

— sets of output channels within a composed component
definition
correctCompositionOut :: *specID* ⇒ *bool*
where
correctCompositionOut x ≡
(*out x*) = (⋃ (*out* ‘ (*subcomponents x*))− (*loc x*))
∧ (*out x*) ∩ (⋃ (*ins* ‘ (*subcomponents x*))) = {}

— Predicate insuring the correct relations between
— sets of local channels within a composed component
definition
correctCompositionLoc :: *specID* ⇒ *bool*
where
correctCompositionLoc x ≡
(*loc x*) = ⋃ (*ins* ‘ (*subcomponents x*))
∩ ⋃ (*out* ‘ (*subcomponents x*))

— If a component is an elementary one (has no subcomponents)
— its set of local channels should be empty
lemma *subcomponents-loc*:
assumes *correctCompositionLoc x*
and *subcomponents x* = {}
shows *loc x* = {}
using *assms* **by** (*simp add*: *correctCompositionLoc-def*)

end

3 Secrecy: Definitions and properties

theory *Secrecy*
imports *Secrecy-types inout ListExtras*
begin

— Encryption, decryption, signature creation and signature verification functions
— For these 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 implementation details
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*

— Axioms on relations between encription and decription keys
axiomatization
EncrDecrKeys :: *Keys* ⇒ *Keys* ⇒ *bool*
where
ExtSign:
EncrDecrKeys K1 K2 ⟶ (*Ext K1* (*Sign K2 E*)) = *E* **and**

DecrEnc:
EncrDecrKeys K1 K2 $\longrightarrow$ (*Decr K2* (*Enc K1 E*)) = *E*

— Set of private keys of a component
consts
specKeys :: *specID* $\Rightarrow$ *Keys set*
— Set of unguessable values used by a component
consts
specSecrets :: *specID* $\Rightarrow$ *Secrets set*

— Join set of private keys and unguessable values used by a component
definition
specKeysSecrets :: *specID* $\Rightarrow$ *KS set*
where
specKeysSecrets C $\equiv$
$\{y\ .\ \exists\ x.\ y = (kKS\ x)\ \wedge (x \in (specKeys\ C))\} \cup$
$\{z\ .\ \exists\ s.\ z = (sKS\ s)\ \wedge (s \in (specSecrets\ C))\}$

— Predicate defining that a list of expression items does not contain
— any private key or unguessable value used by a component
definition
notSpecKeysSecretsExpr :: *specID* $\Rightarrow$ *Expression list* $\Rightarrow$ *bool*
where
notSpecKeysSecretsExpr P e $\equiv$
$(\forall\ x.\ (kE\ x)\ mem\ e \longrightarrow (kKS\ x) \notin specKeysSecrets\ P) \wedge$
$(\forall\ y.\ (sE\ y)\ mem\ e \longrightarrow (sKS\ y) \notin specKeysSecrets\ P)$

— If a component is a composite one, the set of its private keys
— is a union of the subcomponents' sets of the private keys
definition
correctCompositionKeys :: *specID* $\Rightarrow$ *bool*
where
correctCompositionKeys x $\equiv$
subcomponents x $\neq \{\} \longrightarrow$
specKeys x = $\bigcup$ (*specKeys* ‘ (*subcomponents x*))

— If a component is a composite one, the set of its unguessable values
— is a union of the subcomponents' sets of the unguessable values
definition
correctCompositionSecrets :: *specID* $\Rightarrow$ *bool*
where
correctCompositionSecrets x $\equiv$
subcomponents x $\neq \{\} \longrightarrow$
specSecrets x = $\bigcup$ (*specSecrets* ‘ (*subcomponents x*))

— If a component is a composite one, the set of its private keys and
— unguessable values is a union of the corresponding sets of its subcomponents
definition
correctCompositionKS :: *specID* $\Rightarrow$ *bool*

where
correctCompositionKS x ≡
subcomponents x ≠ {} ⟶
specKeysSecrets x = ⋃ (*specKeysSecrets* ‘ (*subcomponents x*))

— Predicate defining set of correctness properties of the component's
— interface and relations on its private keys and unguessable values
definition
correctComponentSecrecy :: *specID* ⇒ *bool*
where
correctComponentSecrecy x ≡
correctCompositionKS x ∧
correctCompositionSecrets x ∧
correctCompositionKeys x ∧
correctCompositionLoc x ∧
correctCompositionIn x ∧
correctCompositionOut x ∧
correctInOutLoc x

— Predicate exprChannel I E defines whether the expression item E can be sent via the channel I
consts
exprChannel :: *chanID* ⇒ *Expression* ⇒ *bool*

— Predicate eoutM sP M E defines whether the component sP may eventually
— output an expression E if there exists a time interval t of
— an output channel which contains this expression E
definition
eout :: *specID* ⇒ *Expression* ⇒ *bool*
where
eout sP E ≡
∃ (*ch* :: *chanID*). ((*ch* ∈ (*out sP*)) ∧ (*exprChannel ch E*))

— Predicate eout sP E defines whether the component sP may eventually
— output an expression E via subset of channels M,
— which is a subset of output channels of sP,
— and if there exists a time interval t of
— an output channel which contains this expression E
definition
eoutM :: *specID* ⇒ *chanID set* ⇒ *Expression* ⇒ *bool*
where
eoutM sP M E ≡
∃ (*ch* :: *chanID*). ((*ch* ∈ (*out sP*)) ∧ (*ch* ∈ *M*) ∧ (*exprChannel ch E*))

— Predicate ineM sP M E defines whether a component sP may eventually
— get an expression E if there exists a time interval t of
— an input stream which contains this expression E
definition
ine :: *specID* ⇒ *Expression* ⇒ *bool*

where
ine sP E ≡
∃ (*ch* :: *chanID*). ((*ch* ∈ (*ins sP*)) ∧ (*exprChannel ch E*))

— Predicate ine sP E defines whether a component sP may eventually
— get an expression E via subset of channels M,
— which is a subset of input channels of sP,
— and if there exists a time interval t of
— an input stream which contains this expression E
definition
ineM :: *specID* ⇒ *chanID set* ⇒ *Expression* ⇒ *bool*
where
ineM sP M E ≡
∃ (*ch* :: *chanID*). ((*ch* ∈ (*ins sP*)) ∧ (*ch* ∈ *M*) ∧ (*exprChannel ch E*))

— This predicate defines whether an input channel ch of a component sP
— is the only one input channel of this component
— via which it may eventually output an expression E
definition
out-exprChannelSingle :: *specID* ⇒ *chanID* ⇒ *Expression* ⇒ *bool*
where
out-exprChannelSingle sP ch E ≡
(*ch* ∈ (*out sP*)) ∧
(*exprChannel ch E*) ∧
(∀ (*x* :: *chanID*) (*t* :: *nat*). ((*x* ∈ (*out sP*)) ∧ (*x* ≠ *ch*) ⟶ ¬ *exprChannel x E*))

— This predicate yields true if only the channels from the set chSet,
— which is a subset of input channels of the component sP,
— may eventually output an expression E
definition
out-exprChannelSet :: *specID* ⇒ *chanID set* ⇒ *Expression* ⇒ *bool*
where
out-exprChannelSet sP chSet E ≡
((∀ (*x* ::*chanID*). ((*x* ∈ *chSet*) ⟶ ((*x* ∈ (*out sP*)) ∧ (*exprChannel x E*))))
∧
(∀ (*x* :: *chanID*). ((*x* ∉ *chSet*) ∧ (*x* ∈ (*out sP*)) ⟶ ¬ *exprChannel x E*)))

— This redicate defines whether
— an input channel ch of a component sP is the only one input channel
— of this component via which it may eventually get an expression E
definition
ine-exprChannelSingle :: *specID* ⇒ *chanID* ⇒ *Expression* ⇒ *bool*
where
ine-exprChannelSingle sP ch E ≡
(*ch* ∈ (*ins sP*)) ∧
(*exprChannel ch E*) ∧
(∀ (*x* :: *chanID*) (*t* :: *nat*). ((*x* ∈ (*ins sP*)) ∧ (*x* ≠ *ch*) ⟶ ¬ *exprChannel x E*))

— This predicate yields true if the component sP may eventually

— get an expression E only via the channels from the set chSet,
— which is a subset of input channels of sP
definition
ine-exprChannelSet :: *specID* ⇒ *chanID set* ⇒ *Expression* ⇒ *bool*
where
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*)))

— If a list of expression items does not contain any private key
— or unguessable value of a component P, then the first element
— of the list is neither a private key nor unguessable value of P
lemma *notSpecKeysSecretsExpr-L1*:
assumes *notSpecKeysSecretsExpr P* (*a* # *l*)
shows *notSpecKeysSecretsExpr P* [*a*]
using *assms* **by** (*simp add*: *notSpecKeysSecretsExpr-def*)

— If a list of expression items does not contain any private key
— or unguessable value of a component P, then this list without its first
— element does not contain them too
lemma *notSpecKeysSecretsExpr-L2*:
assumes *notSpecKeysSecretsExpr P* (*a* # *l*)
shows *notSpecKeysSecretsExpr P l*
using *assms* **by** (*simp add*: *notSpecKeysSecretsExpr-def*)

— If a channel belongs to the set of input channels of a component P
— and does not belong to the set of local channels of the compositon of P and Q
— then it belongs to the set of input channels of this composition
lemma *correctCompositionIn-L1*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ch* ∉ *loc PQ*
and *ch* ∈ *ins P*
shows *ch* ∈ *ins PQ*
using *assms* **by** (*simp add*: *correctCompositionIn-def*)

— If a channel belongs to the set of input channels of the compositon of P and Q
— then it belongs to the set of input channels either of P or of Q
lemma *correctCompositionIn-L2*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ch* ∈ *ins PQ*
shows (*ch* ∈ *ins P*) ∨ (*ch* ∈ *ins Q*)
using *assms* **by** (*simp add*: *correctCompositionIn-def*)

lemma *ineM-L1*:
assumes *ch* ∈ *M*
and *ch* ∈ *ins P*

and *exprChannel ch E*
shows *ineM P M E*
using *assms* **by** (*simp add*: *ineM-def*, *blast*)

lemma *ineM-ine*:
assumes *ineM P M E*
shows *ine P E*
using *assms* **by** (*simp add*: *ineM-def ine-def*, *blast*)

lemma *not-ine-ineM*:
assumes ¬ *ine P E*
shows ¬ *ineM P M E*
using *assms* **by** (*simp add*: *ineM-def ine-def*)

lemma *eoutM-eout*:
assumes *eoutM P M E*
shows *eout P E*
using *assms* **by** (*simp add*: *eoutM-def eout-def*, *blast*)

lemma *not-eout-eoutM*:
assumes ¬ *eout P E*
shows ¬ *eoutM P M E*
using *assms* **by** (*simp add*: *eoutM-def eout-def*)

lemma *correctCompositionKeys-subcomp1*:
assumes *correctCompositionKeys C*
and $x \in$ *subcomponents C*
and $xb \in$ *specKeys C*
shows $\exists\ x \in$ *subcomponents C*. (*xb* $\in$ *specKeys x*)
using *assms* **by** (*simp add*: *correctCompositionKeys-def*, *auto*)

lemma *correctCompositionSecrets-subcomp1*:
assumes *correctCompositionSecrets C*
and $x \in$ *subcomponents C*
and $s \in$ *specSecrets C*
shows $\exists\ x \in$ *subcomponents C*. (*s* $\in$ *specSecrets x*)
using *assms* **by** (*simp add*: *correctCompositionSecrets-def*, *auto*)

lemma *correctCompositionKeys-subcomp2*:
assumes *correctCompositionKeys C*
and $xb \in$ *subcomponents C*
and $xc \in$ *specKeys xb*
shows $xc \in$ *specKeys C*
using *assms* **by** (*simp add*: *correctCompositionKeys-def*, *auto*)

lemma *correctCompositionSecrets-subcomp2*:
assumes *correctCompositionSecrets C*
and $xb \in$ *subcomponents C*
and $xc \in$ *specSecrets xb*

shows *xc* ∈ *specSecrets C*
using *assms* **by** (*simp add*: *correctCompositionSecrets-def*, *auto*)

lemma *correctCompKS-Keys*:
assumes *correctCompositionKS C*
shows *correctCompositionKeys C*
proof (*cases subcomponents C* = {})
assume *subcomponents C* = {}
from *this* **and** *assms* **show** *?thesis*
by (*simp add*: *correctCompositionKeys-def*)
next
assume *subcomponents C* ≠ {}
from *this* **and** *assms* **show** *?thesis*
by (*simp add*: *correctCompositionKS-def*
correctCompositionKeys-def
specKeysSecrets-def, *blast*)
qed

lemma *correctCompKS-Secrets*:
assumes *correctCompositionKS C*
shows *correctCompositionSecrets C*
proof (*cases subcomponents C* = {})
assume *subcomponents C* = {}
from *this* **and** *assms* **show** *?thesis*
by (*simp add*: *correctCompositionSecrets-def*)
next
assume *subcomponents C* ≠ {}
from *this* **and** *assms* **show** *?thesis*
by (*simp add*: *correctCompositionKS-def*
correctCompositionSecrets-def
specKeysSecrets-def, *blast*)
qed

lemma *correctCompKS-KeysSecrets*:
assumes *correctCompositionKeys C*
and *correctCompositionSecrets C*
shows *correctCompositionKS C*
proof (*cases subcomponents C* = {})
assume *subcomponents C* = {}
from *this* **and** *assms* **show** *?thesis*
by (*simp add*: *correctCompositionKS-def*)
next
assume *subcomponents C* ≠ {}
from *this* **and** *assms* **show** *?thesis*
by (*simp add*: *correctCompositionKS-def*
correctCompositionKeys-def
correctCompositionSecrets-def
specKeysSecrets-def, *blast*)
qed

lemma *correctCompositionKS-subcomp1*:
assumes *correctCompositionKS C*
and *h1*:$x \in$ *subcomponents C*
and $xa \in$ *specKeys C*
shows $\exists\ y \in$ *subcomponents C*. ($xa \in$ *specKeys y*)
proof (*cases subcomponents C* = {})
assume *subcomponents C* = {}
from *this* **and** *h1* **show** *?thesis* **by** *simp*
next
assume *subcomponents C* ≠ {}
from *this* **and** *assms* **show** *?thesis*
by (*simp add*: *correctCompositionKS-def specKeysSecrets-def*, *blast*)
qed

lemma *correctCompositionKS-subcomp2*:
assumes *correctCompositionKS C*
and *h1*:$x \in$ *subcomponents C*
and $xa \in$ *specSecrets C*
shows $\exists\ y \in$ *subcomponents C*. $xa \in$ *specSecrets y*
proof (*cases subcomponents C* = {})
assume *subcomponents C* = {}
from *this* **and** *h1* **show** *?thesis* **by** *simp*
next
assume *subcomponents C* ≠ {}
from *this* **and** *assms* **show** *?thesis*
by (*simp add*: *correctCompositionKS-def specKeysSecrets-def*, *blast*)
qed

lemma *correctCompositionKS-subcomp3*:
assumes *correctCompositionKS C*
and $x \in$ *subcomponents C*
and $xa \in$ *specKeys x*
shows $xa \in$ *specKeys C*
using *assms*
by (*simp add*: *correctCompositionKS-def specKeysSecrets-def*, *auto*)

lemma *correctCompositionKS-subcomp4*:
assumes *correctCompositionKS C*
and $x \in$ *subcomponents C*
and $xa \in$ *specSecrets x*
shows $xa \in$ *specSecrets C*
using *assms*
by (*simp add*: *correctCompositionKS-def specKeysSecrets-def*, *auto*)

lemma *correctCompositionKS-PQ*:
assumes *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionKS PQ*
and $ks \in$ *specKeysSecrets PQ*

shows $ks \in specKeysSecrets\ P \vee ks \in specKeysSecrets\ Q$
using *assms* **by** (*simp add*: *correctCompositionKS-def*)

lemma *correctCompositionKS-neg1*:
assumes $subcomponents\ PQ = \{P, Q\}$
and $correctCompositionKS\ PQ$
and $ks \notin specKeysSecrets\ P$
and $ks \notin specKeysSecrets\ Q$
shows $ks \notin specKeysSecrets\ PQ$
using *assms* **by** (*simp add*: *correctCompositionKS-def*)

lemma *correctCompositionKS-negP*:
assumes $subcomponents\ PQ = \{P, Q\}$
and $correctCompositionKS\ PQ$
and $ks \notin specKeysSecrets\ PQ$
shows $ks \notin specKeysSecrets\ P$
using *assms* **by** (*simp add*: *correctCompositionKS-def*)

lemma *correctCompositionKS-negQ*:
assumes $subcomponents\ PQ = \{P, Q\}$
and $correctCompositionKS\ PQ$
and $ks \notin specKeysSecrets\ PQ$
shows $ks \notin specKeysSecrets\ Q$
using *assms* **by** (*simp add*: *correctCompositionKS-def*)

lemma *out-exprChannelSingle-Set*:
assumes $out\text{-}exprChannelSingle\ P\ ch\ E$
shows $out\text{-}exprChannelSet\ P\ \{ch\}\ E$
using *assms*
by (*simp add*: *out-exprChannelSingle-def out-exprChannelSet-def*)

lemma *out-exprChannelSet-Single*:
assumes $out\text{-}exprChannelSet\ P\ \{ch\}\ E$
shows $out\text{-}exprChannelSingle\ P\ ch\ E$
using *assms*
by (*simp add*: *out-exprChannelSingle-def out-exprChannelSet-def*)

lemma *ine-exprChannelSingle-Set*:
assumes $ine\text{-}exprChannelSingle\ P\ ch\ E$
shows $ine\text{-}exprChannelSet\ P\ \{ch\}\ E$
using *assms*
by (*simp add*: *ine-exprChannelSingle-def ine-exprChannelSet-def*)

lemma *ine-exprChannelSet-Single*:
assumes $ine\text{-}exprChannelSet\ P\ \{ch\}\ E$
shows $ine\text{-}exprChannelSingle\ P\ ch\ E$
using *assms*
by (*simp add*: *ine-exprChannelSingle-def ine-exprChannelSet-def*)

lemma *ine-ins-neg1*:
assumes ¬ *ine P m*
and *exprChannel x m*
shows *x* ∉ *ins P*
using *assms* **by** (*simp add*: *ine-def*, *auto*)

theorem *TBtheorem1a*:
assumes *ine PQ E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
shows *ine P E* ∨ *ine Q E*
using *assms*
by (*simp add*: *ine-def correctCompositionIn-def*, *auto*)

theorem *TBtheorem1b*:
assumes *ineM PQ M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
shows *ineM P M E* ∨ *ineM Q M E*
using *assms* **by** (*simp add*: *ineM-def correctCompositionIn-def*, *auto*)

theorem *TBtheorem2a*:
assumes *eout PQ E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
shows *eout P E* ∨ *eout Q E*
using *assms* **by** (*simp add*: *eout-def correctCompositionOut-def*, *auto*)

theorem *TBtheorem2b*:
assumes *eoutM PQ M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
shows *eoutM P M E* ∨ *eoutM Q M E*
using *assms* **by** (*simp add*: *eoutM-def correctCompositionOut-def*, *auto*)

lemma *correctCompositionIn-prop1*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *x* ∈ (*ins PQ*)
shows (*x* ∈ (*ins P*)) ∨ (*x* ∈ (*ins Q*))
using *assms* **by** (*simp add*: *correctCompositionIn-def*)

lemma *correctCompositionOut-prop1*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *x* ∈ (*out PQ*)
shows (*x* ∈ (*out P*)) ∨ (*x* ∈ (*out Q*))
using *assms* **by** (*simp add*: *correctCompositionOut-def*)

theorem *TBtheorem3a*:
assumes ¬ (*ine P E*)
  and ¬ (*ine Q E*)
  and *subcomponents PQ* = {*P*,*Q*}
  and *correctCompositionIn PQ*
shows ¬ (*ine PQ E*)
using *assms* **by** (*simp add*: *ine-def correctCompositionIn-def*, *auto*)

theorem *TBlemma3b*:
assumes *h1*:¬ (*ineM P M E*)
  and *h2*:¬ (*ineM Q M E*)
  and *subPQ*:*subcomponents PQ* = {*P*,*Q*}
  and *cCompI*:*correctCompositionIn PQ*
  and *chM*:*ch* ∈ *M*
  and *chPQ*:*ch* ∈ *ins PQ*
  and *eCh*:*exprChannel ch E*
shows *False*
proof (*cases ch* ∈ *ins P*)
 assume *a1*:*ch* ∈ *ins P*
 from *a1* **and** *chM* **and** *eCh* **have** *ineM P M E* **by** (*simp add*: *ineM-L1*)
 from *this* **and** *h1* **show** *?thesis* **by** *simp*
next
 assume *a2*:*ch* ∉ *ins P*
 from *subPQ* **and** *cCompI* **and** *chPQ* **have** (*ch* ∈ *ins P*) ∨ (*ch* ∈ *ins Q*)
  by (*simp add*: *correctCompositionIn-L2*)
 from *this* **and** *a2* **have** *ch* ∈ *ins Q* **by** *simp*
 from *this* **and** *chM* **and** *eCh* **have** *ineM Q M E* **by** (*simp add*: *ineM-L1*)
 from *this* **and** *h2* **show** *?thesis* **by** *simp*
qed

theorem *TBtheorem3b*:
assumes ¬ (*ineM P M E*)
  and ¬ (*ineM Q M E*)
  and *subcomponents PQ* = {*P*,*Q*}
  and *correctCompositionIn PQ*
shows ¬ (*ineM PQ M E*)
using *assms* **by** (*metis TBtheorem1b*)

theorem *TBtheorem4a-empty*:
assumes (*ine P E*) ∨ (*ine Q E*)
  and *subcomponents PQ* = {*P*,*Q*}
  and *correctCompositionIn PQ*
  and *loc PQ* = {}
shows *ine PQ E*
using *assms* **by** (*simp add*: *ine-def correctCompositionIn-def*, *auto*)

theorem *TBtheorem4a-P*:
assumes *ine P E*
  and *subcomponents PQ* = {*P*,*Q*}

and *correctCompositionIn PQ*
and $\exists$ *ch.* $(ch \in (ins\ P) \wedge exprChannel\ ch\ E \wedge ch \notin (loc\ PQ))$
shows *ine PQ E*
using *assms* **by** (*simp add*: *ine-def correctCompositionIn-def*, *auto*)

theorem *TBtheorem4b-P*:
assumes *ineM P M E*
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
and $\exists$ *ch.* $((ch \in (ins\ Q)) \wedge (exprChannel\ ch\ E) \wedge$
$(ch \notin (loc\ PQ)) \wedge (ch \in M))$
shows *ineM PQ M E*
using *assms* **by** (*simp add*: *ineM-def correctCompositionIn-def*, *auto*)

theorem *TBtheorem4a-PQ*:
assumes $(ine\ P\ E) \vee (ine\ Q\ E)$
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
and $\exists$ *ch.* $(((ch \in (ins\ P)) \vee (ch \in (ins\ Q))) \wedge$
$(exprChannel\ ch\ E) \wedge\ (ch \notin (loc\ PQ)))$
shows *ine PQ E*
using *assms* **by** (*simp add*: *ine-def correctCompositionIn-def*, *auto*)

theorem *TBtheorem4b-PQ*:
assumes $(ineM\ P\ M\ E) \vee (ineM\ Q\ M\ E)$
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
and $\exists$ *ch.* $(((ch \in (ins\ P)) \vee (ch \in (ins\ Q))) \wedge$
$(ch \in M) \wedge (exprChannel\ ch\ E) \wedge\ (ch \notin (loc\ PQ)))$
shows *ineM PQ M E*
using *assms* **by** (*simp add*: *ineM-def correctCompositionIn-def*, *auto*)

theorem *TBtheorem4a-notP1*:
assumes *ine P E*
and $\neg$ *ine Q E*
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
and $\exists$ *ch.* $((ine\text{-}exprChannelSingle\ P\ ch\ E) \wedge (ch \in (loc\ PQ)))$
shows $\neg$ *ine PQ E*
using *assms*
by (*simp add*: *ine-def correctCompositionIn-def*
ine-exprChannelSingle-def, *auto*)

theorem *TBtheorem4b-notP1*:
assumes *ineM P M E*
and $\neg$ *ineM Q M E*
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
and $\exists$ *ch.* $((ine\text{-}exprChannelSingle\ P\ ch\ E) \wedge (ch \in M)$

$\wedge$ ($ch \in (loc\ PQ)$)))
shows $\neg$ *ineM PQ M E*
using *assms*
by (*simp add*: *ineM-def correctCompositionIn-def*
ine-exprChannelSingle-def, *auto*)

theorem *TBtheorem4a-notP2*:
assumes $\neg$ *ine Q E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ine-exprChannelSet P ChSet E*
and $\forall$ (x ::*chanID*). (($x \in ChSet$) $\longrightarrow$ ($x \in (loc\ PQ)$)))
shows $\neg$ *ine PQ E*
using *assms*
by (*simp add*: *ine-def correctCompositionIn-def*
ine-exprChannelSet-def, *auto*)

theorem *TBtheorem4b-notP2*:
assumes $\neg$ *ineM Q M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ine-exprChannelSet P ChSet E*
and $\forall$ (x ::*chanID*). (($x \in ChSet$) $\longrightarrow$ ($x \in (loc\ PQ)$)))
shows $\neg$ *ineM PQ M E*
using *assms*
by (*simp add*: *ineM-def correctCompositionIn-def*
ine-exprChannelSet-def, *auto*)

theorem *TBtheorem4a-notPQ*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ine-exprChannelSet P ChSetP E*
and *ine-exprChannelSet Q ChSetQ E*
and $\forall$ (x ::*chanID*). (($x \in ChSetP$) $\longrightarrow$ ($x \in (loc\ PQ)$)))
and $\forall$ (x ::*chanID*). (($x \in ChSetQ$) $\longrightarrow$ ($x \in (loc\ PQ)$)))
shows $\neg$ *ine PQ E*
using *assms*
by (*simp add*: *ine-def correctCompositionIn-def*
ine-exprChannelSet-def, *auto*)

lemma *ineM-Un1*:
assumes *ineM P A E*
shows *ineM P* (*A Un B*) *E*
using *assms* **by** (*simp add*: *ineM-def*, *auto*)

theorem *TBtheorem4b-notPQ*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ine-exprChannelSet P ChSetP E*

and *ine-exprChannelSet Q ChSetQ E*
and $\forall\ (x :: chanID).\ ((x \in ChSetP) \longrightarrow (x \in (loc\ PQ)))$
and $\forall\ (x :: chanID).\ ((x \in ChSetQ) \longrightarrow (x \in (loc\ PQ)))$
shows $\neg\ ineM\ PQ\ M\ E$
using *assms*
by (*simp add*: *ineM-def correctCompositionIn-def*
ine-exprChannelSet-def, *auto*)

lemma *ine-nonempty-exprChannelSet*:
assumes *ine-exprChannelSet P ChSet E*
and $ChSet \neq \{\}$
shows *ine P E*
using *assms* **by** (*simp add*: *ine-def ine-exprChannelSet-def*, *auto*)

lemma *ine-empty-exprChannelSet*:
assumes *ine-exprChannelSet P ChSet E*
and $ChSet = \{\}$
shows $\neg\ ine\ P\ E$
using *assms* **by** (*simp add*: *ine-def ine-exprChannelSet-def*)

theorem *TBtheorem5a-empty*:
assumes $(eout\ P\ E) \vee (eout\ Q\ E)$
and $subcomponents\ PQ = \{P, Q\}$
and *correctCompositionOut PQ*
and $loc\ PQ = \{\}$
shows *eout PQ E*
using *assms* **by** (*simp add*: *eout-def correctCompositionOut-def*, *auto*)

theorem *TBtheorem45a-P*:
assumes *eout P E*
and $subcomponents\ PQ = \{P, Q\}$
and *correctCompositionOut PQ*
and $\exists\ ch.\ ((ch \in (out\ P)) \wedge (exprChannel\ ch\ E) \wedge$
$(ch \notin (loc\ PQ)))$
shows *eout PQ E*
using *assms* **by** (*simp add*: *eout-def correctCompositionOut-def*, *auto*)

theorem *TBtheore54b-P*:
assumes *eoutM P M E*
and $subcomponents\ PQ = \{P, Q\}$
and *correctCompositionOut PQ*
and $\exists\ ch.\ ((ch \in (out\ Q)) \wedge (exprChannel\ ch\ E) \wedge$
$(ch \notin (loc\ PQ)) \wedge (ch \in M)\ )$
shows *eoutM PQ M E*
using *assms* **by** (*simp add*: *eoutM-def correctCompositionOut-def*, *auto*)

theorem *TBtheorem5a-PQ*:
assumes $(eout\ P\ E) \vee (eout\ Q\ E)$
and $subcomponents\ PQ = \{P, Q\}$

and *correctCompositionOut PQ*
and $\exists$ *ch.* $(((ch \in (out\ P)) \vee (ch \in (out\ Q))) \wedge$
$(exprChannel\ ch\ E) \wedge (ch \notin (loc\ PQ)))$
shows *eout PQ E*
using *assms* **by** (*simp add*: *eout-def correctCompositionOut-def*, *auto*)

theorem *TBtheorem5b-PQ*:
assumes $(eoutM\ P\ M\ E) \vee (eoutM\ Q\ M\ E)$
and *subcomponents PQ* $= \{P,Q\}$
and *correctCompositionOut PQ*
and $\exists$ *ch.* $(((ch \in (out\ P)) \vee (ch \in (out\ Q))) \wedge (ch \in M)$
$\wedge (exprChannel\ ch\ E) \wedge (ch \notin (loc\ PQ)))$
shows *eoutM PQ M E*
using *assms* **by** (*simp add*: *eoutM-def correctCompositionOut-def*, *auto*)

theorem *TBtheorem5a-notP1*:
assumes *eout P E*
and $\neg$ *eout Q E*
and *subcomponents PQ* $= \{P,Q\}$
and *correctCompositionOut PQ*
and $\exists$ *ch.* $((out\text{-}exprChannelSingle\ P\ ch\ E) \wedge (ch \in (loc\ PQ)))$
shows $\neg$ *eout PQ E*
using *assms*
by (*simp add*: *eout-def correctCompositionOut-def*
out-exprChannelSingle-def, *auto*)

theorem *TBtheorem5b-notP1*:
assumes *eoutM P M E*
and $\neg$ *eoutM Q M E*
and *subcomponents PQ* $= \{P,Q\}$
and *correctCompositionOut PQ*
and $\exists$ *ch.* $((out\text{-}exprChannelSingle\ P\ ch\ E) \wedge (ch \in M)$
$\wedge (ch \in (loc\ PQ)))$
shows $\neg$ *eoutM PQ M E*
using *assms*
by (*simp add*: *eoutM-def correctCompositionOut-def*
out-exprChannelSingle-def, *auto*)

theorem *TBtheorem5a-notP2*:
assumes $\neg$ *eout Q E*
and *subcomponents PQ* $= \{P,Q\}$
and *correctCompositionOut PQ*
and *out-exprChannelSet P ChSet E*
and $\forall$ $(x :: chanID).\ ((x \in ChSet) \longrightarrow (x \in (loc\ PQ)))$
shows $\neg$ *eout PQ E*
using *assms*
by (*simp add*: *eout-def correctCompositionOut-def*
out-exprChannelSet-def, *auto*)

theorem *TBtheorem5b-notP2*:
assumes ¬ *eoutM Q M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *out-exprChannelSet P ChSet E*
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSet*) ⟶ (*x* ∈ (*loc PQ*)))
shows ¬ *eoutM PQ M E*
using *assms*
by (*simp add*: *eoutM-def correctCompositionOut-def*
out-exprChannelSet-def, *auto*)

theorem *TBtheorem5a-notPQ*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *out-exprChannelSet P ChSetP E*
and *out-exprChannelSet Q ChSetQ E*
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSetP*) ⟶ (*x* ∈ (*loc PQ*)))
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSetQ*) ⟶ (*x* ∈ (*loc PQ*)))
shows ¬ *eout PQ E*
using *assms*
by (*simp add*: *eout-def correctCompositionOut-def*
out-exprChannelSet-def, *auto*)

theorem *TBtheorem5b-notPQ*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *out-exprChannelSet P ChSetP E*
and *out-exprChannelSet Q ChSetQ E*
and *M* = *ChSetP* ∪ *ChSetQ*
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSetP*) ⟶ (*x* ∈ (*loc PQ*)))
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSetQ*) ⟶ (*x* ∈ (*loc PQ*)))
shows ¬ *eoutM PQ M E*
using *assms*
by (*simp add*: *eoutM-def correctCompositionOut-def*
out-exprChannelSet-def, *auto*)

end

4 Local Secrets of a component

theory *CompLocalSecrets*
imports *Secrecy*
begin

— Set of local secrets: the set of secrets which does not belong to
— the set of private keys and unguessable values, but are transmitted
— via local channels or belongs to the local secrets of its subcomponents
axiomatization
LocalSecrets :: *specID* ⇒ *KS set*

where
LocalSecretsDef:
LocalSecrets A =
{(*m* :: *KS*). *m* ∉ *specKeysSecrets A* ∧
((∃ *x y*. ((*x* ∈ *loc A*) ∧ *m* = (*kKS y*) ∧ (*exprChannel x* (*kE y*))))
|(∃ *x z*. ((*x* ∈ *loc A*) ∧ *m* = (*sKS z*) ∧ (*exprChannel x* (*sE z*)))))}
∪ (⋃ (*LocalSecrets* ‘ (*subcomponents A*)))

lemma *LocalSecretsComposition1*:
assumes *ls* ∈ *LocalSecrets P*
and *subcomponents PQ* = {*P*, *Q*}
shows *ls* ∈ *LocalSecrets PQ*
using *assms* **by** (*simp* (*no-asm*) *only*: *LocalSecretsDef*, *auto*)

lemma *LocalSecretsComposition-exprChannel-k*:
assumes *exprChannel x* (*kE Keys*)
and ¬ *ine P* (*kE Keys*)
and ¬ *ine Q* (*kE Keys*)
and ¬ (*x* ∉ *ins P* ∧ *x* ∉ *ins Q*)
shows *False*
using *assms* **by** (*metis ine-def*)

lemma *LocalSecretsComposition-exprChannel-s*:
assumes *exprChannel x* (*sE Secrets*)
and ¬ *ine P* (*sE Secrets*)
and ¬ *ine Q* (*sE Secrets*)
and ¬ (*x* ∉ *ins P* ∧ *x* ∉ *ins Q*)
shows *False*
using *assms* **by** (*metis ine-ins-neg1*)

lemma *LocalSecretsComposition-neg1-k*:
assumes *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionLoc PQ*
and ¬ *ine P* (*kE Keys*)
and ¬ *ine Q* (*kE Keys*)
and *kKS Keys* ∉ *LocalSecrets P*
and *kKS Keys* ∉ *LocalSecrets Q*
shows *kKS Keys* ∉ *LocalSecrets PQ*
proof −
from *assms* **show** *?thesis*
apply (*simp* (*no-asm*) *only*: *LocalSecretsDef*,
simp add: *correctCompositionLoc-def*, *clarify*)
by (*rule LocalSecretsComposition-exprChannel-k*, *auto*)
qed

lemma *LocalSecretsComposition-neg-k*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionLoc PQ*
and *correctCompositionKS PQ*

and $(kKS\ m) \notin specKeysSecrets\ P$
and $(kKS\ m) \notin specKeysSecrets\ Q$
and $\neg\ ine\ P\ (kE\ m)$
and $\neg\ ine\ Q\ (kE\ m)$
and $(kKS\ m) \notin ((LocalSecrets\ P) \cup (LocalSecrets\ Q))$
shows $(kKS\ m) \notin (LocalSecrets\ PQ)$
proof −
from *assms* **show** *?thesis*
apply (*simp* (*no-asm*) *only*: *LocalSecretsDef*,
simp add: *correctCompositionLoc-def*, *clarify*)
by (*rule LocalSecretsComposition-exprChannel-k*, *auto*)
qed

lemma *LocalSecretsComposition-neg-s*:
assumes *subPQ*:*subcomponents PQ* = {*P*,*Q*}
and *cCompLoc*:*correctCompositionLoc PQ*
and *cCompKS*:*correctCompositionKS PQ*
and *notKSP*:$(sKS\ m) \notin specKeysSecrets\ P$
and *notKSQ*:$(sKS\ m) \notin specKeysSecrets\ Q$
and $\neg\ ine\ P\ (sE\ m)$
and $\neg\ ine\ Q\ (sE\ m)$
and *notLocSeqPQ*:$(sKS\ m) \notin ((LocalSecrets\ P) \cup (LocalSecrets\ Q))$
shows $(sKS\ m) \notin (LocalSecrets\ PQ)$
proof −
from *subPQ* **and** *cCompKS* **and** *notKSP* **and** *notKSQ*
have *sg1*:$sKS\ m \notin specKeysSecrets\ PQ$
by (*simp add*: *correctCompositionKS-neg1*)
from *subPQ* **and** *cCompLoc* **and** *notLocSeqPQ* **have** *sg2*:
$sKS\ m \notin \bigcup (LocalSecrets\ `\ subcomponents\ PQ)$
by *simp*
from *sg1* **and** *sg2* **and** *assms* **show** *?thesis*
apply (*simp* (*no-asm*) *only*: *LocalSecretsDef*,
simp add: *correctCompositionLoc-def*, *clarify*)
by (*rule LocalSecretsComposition-exprChannel-s*, *auto*)
qed

lemma *LocalSecretsComposition-neg*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionLoc PQ*
and *correctCompositionKS PQ*
and $ks \notin specKeysSecrets\ P$
and $ks \notin specKeysSecrets\ Q$
and *h1*:$\forall\ m.\ ks = kKS\ m \longrightarrow (\neg\ ine\ P\ (kE\ m) \wedge \neg\ ine\ Q\ (kE\ m))$
and *h2*:$\forall\ m.\ ks = sKS\ m \longrightarrow (\neg\ ine\ P\ (sE\ m) \wedge \neg\ ine\ Q\ (sE\ m))$
and $ks \notin ((LocalSecrets\ P) \cup (LocalSecrets\ Q))$
shows $ks \notin (LocalSecrets\ PQ)$
proof (*cases ks*)
fix *m*
assume *a1*:$ks = kKS\ m$

from *this* **and** *h1* **have** ¬ *ine P* (*kE m*) ∧ ¬ *ine Q* (*kE m*) **by** *simp*
from *this* **and** *a1* **and** *assms* **show** *?thesis*
by (*simp add*: *LocalSecretsComposition-neg-k*)
next
fix *m*
assume *a2*:*ks* = *sKS m*
from *this* **and** *h2* **have** ¬ *ine P* (*sE m*) ∧ ¬ *ine Q* (*sE m*) **by** *simp*
from *this* **and** *a2* **and** *assms* **show** *?thesis*
by (*simp add*: *LocalSecretsComposition-neg-s*)
qed

lemma *LocalSecretsComposition-neg1-s*:
assumes *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionLoc PQ*
and ¬ *ine P* (*sE s*)
and ¬ *ine Q* (*sE s*)
and *sKS s* ∉ *LocalSecrets P*
and *sKS s* ∉ *LocalSecrets Q*
shows *sKS s* ∉ *LocalSecrets PQ*
proof −
from *assms* **have**
sKS s ∉ ⋃ (*LocalSecrets* ‘ *subcomponents PQ*)
by *simp*
from *assms* **and** *this* **show** *?thesis*
apply (*simp* (*no-asm*) *only*: *LocalSecretsDef*,
simp add: *correctCompositionLoc-def*, *clarify*)
by (*rule LocalSecretsComposition-exprChannel-s*, *auto*)
qed

lemma *LocalSecretsComposition-neg1*:
assumes *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionLoc PQ*
and *h1*:∀ *m*. *ks* = *kKS m* ⟶ (¬ *ine P* (*kE m*) ∧ ¬ *ine Q* (*kE m*))
and *h2*:∀ *m*. *ks* = *sKS m* ⟶ (¬ *ine P* (*sE m*) ∧ ¬ *ine Q* (*sE m*))
and *ks* ∉ *LocalSecrets P*
and *ks* ∉ *LocalSecrets Q*
shows *ks* ∉ *LocalSecrets PQ*
proof (*cases ks*)
fix *m*
assume *a1*:*ks* = *kKS m*
from *this* **and** *h1* **have** ¬ *ine P* (*kE m*) ∧ ¬ *ine Q* (*kE m*) **by** *simp*
from *this* **and** *a1* **and** *assms* **show** *?thesis*
by (*simp add*: *LocalSecretsComposition-neg1-k*)
next
fix *m*
assume *a2*:*ks* = *sKS m*
from *this* **and** *h2* **have** ¬ *ine P* (*sE m*) ∧ ¬ *ine Q* (*sE m*) **by** *simp*
from *this* **and** *a2* **and** *assms* **show** *?thesis*
by (*simp add*: *LocalSecretsComposition-neg1-s*)

qed

lemma *LocalSecretsComposition-ine1-k*:
assumes *kKS k* ∈ *LocalSecrets PQ*
and *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionLoc PQ*
and ¬ *ine Q* (*kE k*)
and *kKS k* ∉ *LocalSecrets P*
and *kKS k* ∉ *LocalSecrets Q*
shows *ine P* (*kE k*)
using *assms* **by** (*metis LocalSecretsComposition-neg1-k*)

lemma *LocalSecretsComposition-ine1-s*:
assumes *sKS s* ∈ *LocalSecrets PQ*
and *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionLoc PQ*
and ¬ *ine Q* (*sE s*)
and *sKS s* ∉ *LocalSecrets P*
and *sKS s* ∉ *LocalSecrets Q*
shows *ine P* (*sE s*)
using *assms* **by** (*metis LocalSecretsComposition-neg1-s*)

lemma *LocalSecretsComposition-ine2-k*:
assumes *kKS k* ∈ *LocalSecrets PQ*
and *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionLoc PQ*
and ¬ *ine P* (*kE k*)
and *kKS k* ∉ *LocalSecrets P*
and *kKS k* ∉ *LocalSecrets Q*
shows *ine Q* (*kE k*)
using *assms* **by** (*metis LocalSecretsComposition-ine1-k*)

lemma *LocalSecretsComposition-ine2-s*:
assumes *sKS s* ∈ *LocalSecrets PQ*
and *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionLoc PQ*
and ¬ *ine P* (*sE s*)
and *sKS s* ∉ *LocalSecrets P*
and *sKS s* ∉ *LocalSecrets Q*
shows *ine Q* (*sE s*)
using *assms* **by** (*metis LocalSecretsComposition-ine1-s*)

lemma *LocalSecretsComposition-neg-loc-k*:
assumes *kKS key* ∉ *LocalSecrets P*
and *exprChannel ch* (*kE key*)
and *kKS key* ∉ *specKeysSecrets P*
shows *ch* ∉ *loc P*
using *assms* **by** (*simp only*: *LocalSecretsDef*, *auto*)

lemma *LocalSecretsComposition-neg-loc-s*:
assumes *sKS secret* $\notin$ *LocalSecrets P*
and *exprChannel ch* (*sE secret*)
and *sKS secret* $\notin$ *specKeysSecrets P*
shows *ch* $\notin$ *loc P*
using *assms* **by** (*simp only*: *LocalSecretsDef*, *auto*)

lemma *correctCompositionKS-exprChannel-k-P*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *kKS key* $\notin$ *LocalSecrets PQ*
and *ch* $\in$ *ins P*
and *exprChannel ch* (*kE key*)
and *kKS key* $\notin$ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows *ch* $\in$ *ins PQ* $\wedge$ *exprChannel ch* (*kE key*)
using *assms*
by (*metis LocalSecretsComposition-neg-loc-k correctCompositionIn-L1*)

lemma *correctCompositionKS-exprChannel-k-Pex*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *kKS key* $\notin$ *LocalSecrets PQ*
and *ch* $\in$ *ins P*
and *exprChannel ch* (*kE key*)
and *kKS key* $\notin$ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows $\exists$ *ch*. *ch* $\in$ *ins PQ* $\wedge$ *exprChannel ch* (*kE key*)
using *assms*
by (*metis correctCompositionKS-exprChannel-k-P*)

lemma *correctCompositionKS-exprChannel-k-Q*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *kKS key* $\notin$ *LocalSecrets PQ*
and *ch* $\in$ *ins Q*
and *h1*:*exprChannel ch* (*kE key*)
and *kKS key* $\notin$ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows *ch* $\in$ *ins PQ* $\wedge$ *exprChannel ch* (*kE key*)
proof −
from *assms* **have** *ch* $\notin$ *loc PQ*
by (*simp add*: *LocalSecretsComposition-neg-loc-k*)
from *this* **and** *assms* **have** *ch* $\in$ *ins PQ*
by (*simp add*: *correctCompositionIn-def*)
from *this* **and** *h1* **show** *?thesis* **by** *simp*
qed

lemma *correctCompositionKS-exprChannel-k-Qex*:

assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *kKS key* ∉ *LocalSecrets PQ*
and *ch* ∈ *ins Q*
and *exprChannel ch* (*kE key*)
and *kKS key* ∉ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows ∃ *ch*. *ch* ∈ *ins PQ* ∧ *exprChannel ch* (*kE key*)
using *assms*
by (*metis correctCompositionKS-exprChannel-k-Q*)

lemma *correctCompositionKS-exprChannel-s-P*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *sKS secret* ∉ *LocalSecrets PQ*
and *ch* ∈ *ins P*
and *exprChannel ch* (*sE secret*)
and *sKS secret* ∉ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows *ch* ∈ *ins PQ* ∧ *exprChannel ch* (*sE secret*)
using *assms*
by (*metis LocalSecretsComposition-neg-loc-s correctCompositionIn-L1*)

lemma *correctCompositionKS-exprChannel-s-Pex*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *sKS secret* ∉ *LocalSecrets PQ*
and *ch* ∈ *ins P*
and *exprChannel ch* (*sE secret*)
and *sKS secret* ∉ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows ∃ *ch*. *ch* ∈ *ins PQ* ∧ *exprChannel ch* (*sE secret*)
using *assms*
by (*metis correctCompositionKS-exprChannel-s-P*)

lemma *correctCompositionKS-exprChannel-s-Q*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *sKS secret* ∉ *LocalSecrets PQ*
and *ch* ∈ *ins Q*
and *h1*:*exprChannel ch* (*sE secret*)
and *sKS secret* ∉ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows *ch* ∈ *ins PQ* ∧ *exprChannel ch* (*sE secret*)
proof −
from *assms* **have** *ch* ∉ *loc PQ*
by (*simp add*: *LocalSecretsComposition-neg-loc-s*)
from *this* **and** *assms* **have** *ch* ∈ *ins PQ*
by (*simp add*: *correctCompositionIn-def*)

from *this* **and** *h1* **show** *?thesis* **by** *simp*
qed

lemma *correctCompositionKS-exprChannel-s-Qex*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *sKS secret* ∉ *LocalSecrets PQ*
and *ch* ∈ *ins Q*
and *exprChannel ch* (*sE secret*)
and *sKS secret* ∉ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows ∃ *ch*. *ch* ∈ *ins PQ* ∧ *exprChannel ch* (*sE secret*)
using *assms*
by (*metis correctCompositionKS-exprChannel-s-Q*)

end

5 Knowledge of Keys and Secrets

theory *KnowledgeKeysSecrets*
imports *CompLocalSecrets*
begin
An component A knows a secret m (or some secret expression m) that does not belong to its local sectrets , if

- *A may eventually get the secret m,*
- *m belongs to the set* LS_A *of its local secrets,*
- *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 lists of secrets* m_1 *and* 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.*

primrec
know :: *specID* ⇒ *KS* ⇒ *bool*
where
know A (*kKS m*) =
((*ine A* (*kE m*)) ∨ ((*kKS m*) ∈ (*LocalSecrets A*))) |
know A (*sKS m*) =
((*ine A* (*sE m*)) ∨ ((*sKS m*) ∈ (*LocalSecrets A*)))

axiomatization
  knows :: *specID* ⇒ *Expression list* ⇒ *bool*
where
knows-emptyexpression:
  knows C [] = *True* **and**
know1k:
  knows C [*KS2Expression* (*kKS m1*)] = *know C* (*kKS m1*) **and**
know1s:
  knows C [*KS2Expression* (*sKS m2*)] = *know C* (*sKS m2*) **and**
knows2a:
  knows A (*e1* @ *e*) ⟶ *knows A e* **and**
knows2b:
  knows A (*e* @ *e1*) ⟶ *knows A e* **and**
knows3:
  (*knows A e1*) ∧ (*knows A e2*) ⟶ *knows A* (*e1* @ *e2*) **and**
knows4:
  (*IncrDecrKeys k1 k2*) ∧ (*know A* (*kKS k2*)) ∧ (*knows A* (*Enc k1 e*))
   ⟶ *knows A e*
and
knows5:
  (*IncrDecrKeys k1 k2*) ∧ (*know A* (*kKS k1*)) ∧ (*knows A* (*Sign k2 e*))
   ⟶ *knows A e*
and
knows6:
  (*know A* (*kKS k*)) ∧ (*knows A e1*) ⟶ *knows A* (*Enc k e1*)
and
knows7:
  (*know A* (*kKS k*)) ∧ (*knows A e1*) ⟶ *knows A* (*Sign k e1*)

primrec *eoutKnowCorrect* :: *specID* ⇒ *KS* ⇒ *bool*
where
eout-know-k:
  eoutKnowCorrect C (*kKS m*) =
  ((*eout C* (*kE m*)) ⟷ (*m* ∈ (*specKeys C*) ∨ (*know C* (*kKS m*)))) |
eout-know-s:
  eoutKnowCorrect C (*sKS m*) =
  ((*eout C* (*sE m*)) ⟷ (*m* ∈ (*specSecrets C*) ∨ (*know C* (*sKS m*))))

definition *eoutKnowsECorrect* :: *specID* ⇒ *Expression* ⇒ *bool*
where
  eoutKnowsECorrect C e ≡
   ((*eout C e*) ⟷
   ((∃ *k*. *e* = (*kE k*) ∧ (*k* ∈ *specKeys C*)) ∨
    (∃ *s*. *e* = (*sE s*) ∧ (*s* ∈ *specSecrets C*)) ∨
    (*knows C* [*e*])))

lemma *eoutKnowCorrect-L1k*:
assumes *eoutKnowCorrect C* (*kKS m*)

and *eout C* (*kE m*)
shows *m* ∈ (*specKeys C*) ∨ (*know C* (*kKS m*))
using *assms* **by** (*metis eout-know-k*)

lemma *eoutKnowCorrect-L1s*:
assumes *eoutKnowCorrect C* (*sKS m*)
and *eout C* (*sE m*)
shows *m* ∈ (*specSecrets C*) ∨ (*know C* (*sKS m*))
using *assms* **by** (*metis eout-know-s*)

lemma *eoutKnowsECorrect-L1*:
assumes *eoutKnowsECorrect C e*
and *eout C e*
shows (∃ *k*. *e* = (*kE k*) ∧ (*k* ∈ *specKeys C*)) ∨
(∃ *s*. *e* = (*sE s*) ∧ (*s* ∈ *specSecrets C*)) ∨
(*knows C* [*e*])
using *assms* **by** (*metis eoutKnowsECorrect-def*)

lemma *know2knows-k*:
assumes *know A* (*kKS m*)
shows *knows A* [*kE m*]
using *assms*
by (*metis KS2Expression.simps*(*1*) *know1k*)

lemma *knows2know-k*:
assumes *knows A* [*kE m*]
shows *know A* (*kKS m*)
using *assms*
by (*metis KS2Expression.simps*(*1*) *know1k*)

lemma *know2knowsPQ-k*:
assumes *know P* (*kKS m*) ∨ *know Q* (*kKS m*)
shows *knows P* [*kE m*] ∨ *knows Q* [*kE m*]
using *assms* **by** (*metis know2knows-k*)

lemma *knows2knowPQ-k*:
assumes *knows P* [*kE m*] ∨ *knows Q* [*kE m*]
shows *know P* (*kKS m*) ∨ *know Q* (*kKS m*)
using *assms* **by** (*metis knows2know-k*)

lemma *knows1k*:
know A (*kKS m*) = *knows A* [*kE m*]
by (*metis know2knows-k knows2know-k*)

lemma *know2knows-neg-k*:
assumes ¬ *know A* (*kKS m*)
shows ¬ *knows A* [*kE m*]
using *assms* **by** (*metis knows1k*)

lemma *knows2know-neg-k*:
assumes ¬ *knows A* [*kE m*]
shows ¬ *know A* (*kKS m*)
using *assms* **by** (*metis know2knowsPQ-k*)

lemma *know2knows-s*:
assumes *know A* (*sKS m*)
shows *knows A* [*sE m*]
using *assms*
by (*metis KS2Expression.simps*(*2*) *know1s*)

lemma *knows2know-s*:
assumes *knows A* [*sE m*]
shows *know A* (*sKS m*)
using *assms*
by (*metis KS2Expression.simps*(*2*) *know1s*)

lemma *know2knowsPQ-s*:
assumes *know P* (*sKS m*) ∨ *know Q* (*sKS m*)
shows *knows P* [*sE m*] ∨ *knows Q* [*sE m*]
using *assms* **by** (*metis know2knows-s*)

lemma *knows2knowPQ-s*:
assumes *knows P* [*sE m*] ∨ *knows Q* [*sE m*]
shows *know P* (*sKS m*) ∨ *know Q* (*sKS m*)
using *assms* **by** (*metis knows2know-s*)

lemma *knows1s*:
know A (*sKS m*) = *knows A* [*sE m*]
by (*metis know2knows-s knows2know-s*)

lemma *know2knows-neg-s*:
assumes ¬ *know A* (*sKS m*)
shows ¬ *knows A* [*sE m*]
using *assms* **by** (*metis knows2know-s*)

lemma *knows2know-neg-s*:
assumes ¬ *knows A* [*sE m*]
shows ¬ *know A* (*sKS m*)
using *assms* **by** (*metis know2knows-s*)

lemma *knows2*:
assumes *e2* = *e1* @ *e* ∨ *e2* = *e* @ *e1*
and *knows A e2*
shows *knows A e*
using *assms* **by** (*metis knows2a knows2b*)

lemma *correctCompositionInLoc-exprChannel*:
assumes *subcomponents PQ* = {*P*, *Q*}

and *correctCompositionIn PQ*
and *ch* : *ins P*
and *exprChannel ch m*
and $\forall\ x.\ x \in ins\ PQ \longrightarrow \neg\ exprChannel\ x\ m$
shows *ch* : *loc PQ*
using *assms* **by** (*simp add*: *correctCompositionIn-def*, *auto*)

lemma *eout-know-nonKS-k*:
assumes $m \notin specKeys\ A$
and *eout A* (*kE m*)
and *eoutKnowCorrect A* (*kKS m*)
shows *know A* (*kKS m*)
using *assms* **by** (*metis eoutKnowCorrect-L1k*)

lemma *eout-know-nonKS-s*:
assumes $m \notin specSecrets\ A$
and *eout A* (*sE m*)
and *eoutKnowCorrect A* (*sKS m*)
shows *know A* (*sKS m*)
using *assms* **by** (*metis eoutKnowCorrect-L1s*)

lemma *not-know-k-not-ine*:
assumes $\neg$ *know A* (*kKS m*)
shows $\neg$ *ine A* (*kE m*)
using *assms* **by** *simp*

lemma *not-know-s-not-ine*:
assumes $\neg$ *know A* (*sKS m*)
shows $\neg$ *ine A* (*sE m*)
using *assms* **by** *simp*

lemma *not-know-k-not-eout*:
assumes $m \notin specKeys\ A$
and $\neg$ *know A* (*kKS m*)
and *eoutKnowCorrect A* (*kKS m*)
shows $\neg$ *eout A* (*kE m*)
using *assms* **by** (*metis eout-know-k*)

lemma *not-know-s-not-eout*:
assumes $m \notin specSecrets\ A$
and $\neg$ *know A* (*sKS m*)
and *eoutKnowCorrect A* (*sKS m*)
shows $\neg$ *eout A* (*sE m*)
using *assms* **by** (*metis eout-know-nonKS-s*)

lemma *adv-not-know1*:
assumes *out P* $\subseteq$ *ins A*
and $\neg$ *know A* (*kKS m*)
shows $\neg$ *eout P* (*kE m*)

using *assms*
by (*metis* (*full-types*) *eout-def ine-ins-neg1 not-know-k-not-ine rev-subsetD*)

lemma *adv-not-know2*:
assumes *out P* ⊆ *ins A*
  and ¬ *know A* (*sKS m*)
shows ¬ *eout P* (*sE m*)
using *assms*
by (*metis* (*full-types*) *eout-def ine-ins-neg1 not-know-s-not-ine rev-subsetD*)

lemma *LocalSecrets-L1*:
assumes (*kKS*) *key* ∈ *LocalSecrets P*
  and (*kKS key*) ∉ ⋃(*LocalSecrets ‘ subcomponents P*)
shows *kKS key* ∉ *specKeysSecrets P*
using *assms* **by** (*simp only*: *LocalSecretsDef*, *auto*)

lemma *LocalSecrets-L2*:
assumes *kKS key* ∈ *LocalSecrets P*
  and *kKS key* ∈ *specKeysSecrets P*
shows *kKS key* ∈ ⋃(*LocalSecrets ‘ subcomponents P*)
using *assms* **by** (*simp only*: *LocalSecretsDef*, *auto*)

lemma *know-composition1*:
assumes *notKSP*:*m* ∉ *specKeysSecrets P*
  and *notKSQ*:*m* ∉ *specKeysSecrets Q*
  and *know P m*
  and *subPQ*:*subcomponents PQ* = {*P*,*Q*}
  and *cCompI*:*correctCompositionIn PQ*
  and *cCompKS*:*correctCompositionKS PQ*
shows *know PQ m*
proof (*cases m*)
 fix *key*
 assume *a1*:*m* = *kKS key*
 show *?thesis*
 proof (*cases ine P* (*kE key*))
  assume *a11*:*ine P* (*kE key*)
  from *this* **have** *a11ext*:*ine P* (*kE key*) | *ine Q* (*kE key*) **by** *simp*
  from *subPQ* **and** *cCompKS* **and** *notKSP* **and** *notKSQ*
   have *m* ∉ *specKeysSecrets PQ*
   by (*rule correctCompositionKS-neg1*)
  from *this* **and** *a1* **have** *sg1*:*kKS key* ∉ *specKeysSecrets PQ* **by** *simp*
  from *a1* **and** *a11ext* **and** *cCompKS* **show** *?thesis*
  proof (*cases loc PQ* = {})
   assume *a11locE*:*loc PQ* = {}
   from *a11ext* **and** *subPQ* **and** *cCompI* **and** *a11locE* **have** *ine PQ* (*kE key*)
    by (*rule TBtheorem4a-empty*)
   from *this* **and** *a1* **show** *?thesis* **by** *auto*
  next
   assume *a11locNE*:*loc PQ* ≠ {}

```
      from a1 and a11 and sg1 and assms show ?thesis
        apply (simp add: ine-def, auto)
        by (simp add: correctCompositionKS-exprChannel-k-Pex)
    qed
   next
    assume a12:¬ ine P (kE key)
    from this and a1 and assms show ?thesis
      by (auto, simp add:  LocalSecretsComposition1)
   qed
 next
  fix secret
  assume a2:m = sKS secret
  show ?thesis
  proof (cases  ine P (sE secret))
    assume a21:ine P (sE secret)
    from this have a21ext:ine P (sE secret) | ine Q (sE secret) by simp
     from subPQ and cCompKS and notKSP and notKSQ have m ∉ specK-
eysSecrets PQ
      by (rule correctCompositionKS-neg1)
    from this and a2 have sg2:sKS secret ∉ specKeysSecrets PQ by simp
    from a2 and a21ext and cCompKS  show ?thesis
    proof (cases loc PQ = {})
      assume a21locE:loc PQ = {}
     from a21ext and subPQ and cCompI and a21locE have ine PQ (sE secret)

        by (rule TBtheorem4a-empty)
      from this and a2 show ?thesis by auto
    next
      assume a21locNE:loc PQ ≠ {}
      from a2 and a21 and sg2 and assms show ?thesis
        apply (simp add: ine-def, auto)
        by (simp add: correctCompositionKS-exprChannel-s-Pex)
    qed
   next
    assume a12:¬ ine P (sE secret)
    from this and a2 and assms show ?thesis
    by (metis LocalSecretsComposition1 know.simps(2))
   qed
qed

lemma know-composition2:
assumes m ∉ specKeysSecrets P
      and m ∉ specKeysSecrets Q
      and know Q m
      and subcomponents PQ = {P,Q}
      and correctCompositionIn PQ
      and correctCompositionKS PQ
shows    know PQ m
using assms by (metis insert-commute know-composition1)
```

lemma *know-composition*:
assumes $m \notin$ *specKeysSecrets P*
and $m \notin$ *specKeysSecrets Q*
and *know P m* $\vee$ *know Q m*
and *subcomponents PQ* $= \{P,Q\}$
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *know PQ m*
using *assms* **by** (*metis know-composition1 know-composition2*)

theorem *know-composition-neg-ine-k*:
assumes $\neg$ *know P* (*kKS key*)
and $\neg$ *know Q* (*kKS key*)
and *subcomponents PQ* $= \{P,Q\}$
and *correctCompositionIn PQ*
shows $\neg$ (*ine PQ* (*kE key*))
using *assms* **by** (*metis TBtheorem3a not-know-k-not-ine*)

theorem *know-composition-neg-ine-s*:
assumes $\neg$ *know P* (*sKS secret*)
and $\neg$ *know Q* (*sKS secret*)
and *subcomponents PQ* $= \{P,Q\}$
and *correctCompositionIn PQ*
shows $\neg$ (*ine PQ* (*sE secret*))
using *assms* **by** (*metis TBtheorem3a not-know-s-not-ine*)

lemma *know-composition-neg1*:
assumes *notknowP*:$\neg$ *know P m*
and *notknowQ*:$\neg$ *know Q m*
and *subPQ*:*subcomponents PQ* $= \{P,Q\}$
and *cCompLoc*:*correctCompositionLoc PQ*
and *cCompI*:*correctCompositionIn PQ*
shows $\neg$ *know PQ m*
proof (*cases m*)
fix *key*
assume *a1*:$m =$ *kKS key*
from *notknowP* **and** *a1* **have** *sg1*:$\neg$ *know P* (*kKS key*) **by** *simp*
then have *sg1a*:$\neg$ *ine P* (*kE key*) **by** *simp*
from *sg1* **have** *sg1b*:*kKS key* $\notin$ *LocalSecrets P* **by** *simp*
from *notknowQ* **and** *a1* **have** *sg2*:$\neg$ *know Q* (*kKS key*) **by** *simp*
then have *sg2a*:$\neg$ *ine Q* (*kE key*) **by** *simp*
from *sg2* **have** *sg2b*:*kKS key* $\notin$ *LocalSecrets Q* **by** *simp*
from *sg1* **and** *sg2* **and** *subPQ* **and** *cCompI* **have** *sg3*:$\neg$ *ine PQ* (*kE key*)
by (*rule know-composition-neg-ine-k*)
from *subPQ* **and** *cCompLoc* **and** *sg1a* **and** *sg2a* **and** *sg1b* **and** *sg2b* **have** *sg4*:
kKS key $\notin$ *LocalSecrets PQ*
by (*rule LocalSecretsComposition-neg1-k*)
from *sg3* **and** *sg4* **and** *a1* **show** *?thesis* **by** *simp*

```
next
  fix secret
  assume a2:m = sKS secret
  from notknowP and a2 have sg1:¬ know P (sKS secret) by simp
  then have sg1a:¬ ine P (sE secret) by simp
  from sg1 have sg1b:sKS secret ∉ LocalSecrets P by simp
  from notknowQ and a2 have sg2:¬ know Q (sKS secret) by simp
  then have sg2a:¬ ine Q (sE secret) by simp
  from sg2 have sg2b:sKS secret ∉ LocalSecrets Q by simp
  from sg1 and sg2 and subPQ and cCompI have sg3:¬ ine PQ (sE secret)
    by (rule know-composition-neg-ine-s)
  from subPQ and cCompLoc and sg1a and sg2a and sg1b and sg2b have sg4:
  sKS secret ∉ LocalSecrets PQ
    by (rule LocalSecretsComposition-neg1-s)
  from sg3 and sg4 and a2 show ?thesis by simp
qed

lemma know-decomposition:
assumes knowPQ:know PQ m
       and subPQ:subcomponents PQ = {P,Q}
       and cCompI:correctCompositionIn PQ
       and cCompLoc:correctCompositionLoc PQ
shows know P m ∨ know Q m
proof (cases m)
  fix key
  assume a1:m = kKS key
  from this show ?thesis
  proof (cases ine PQ (kE key))
    assume a11:ine PQ (kE key)
    from this and subPQ and cCompI and a1 have
     ine P (kE key)  ∨ ine Q (kE key)
      by (simp add: TBtheorem1a)
    from this and a1 show ?thesis by auto
  next
    assume a12:¬ ine PQ (kE key)
    from this and knowPQ and a1 have sg2:kKS key ∈ LocalSecrets PQ by auto
    show ?thesis
    proof (cases know Q m)
      assume know Q m
      from this show ?thesis by simp
    next
      assume not-knowQm:¬ know Q m
      from not-knowQm and a1 have sg3a:¬ ine Q (kE key) by simp
      from not-knowQm and a1 have sg3b:kKS key ∉ LocalSecrets Q by simp
      show ?thesis
      proof (cases kKS key ∈ LocalSecrets P)
        assume kKS key ∈ LocalSecrets P
        from this and a1 show ?thesis by simp
      next
```

```
      assume kKS key ∉ LocalSecrets P
       from sg2 and subPQ and cCompLoc and sg3a and this and sg3b have
ine P (kE key)
        by (simp add: LocalSecretsComposition-ine1-k)
      from this and a1 show ?thesis by simp
     qed
   qed
  qed
next
  fix secret
  assume a2:m = sKS secret
  from this show ?thesis
  proof (cases ine PQ (sE secret))
   assume a21:ine PQ (sE secret)
   from this and subPQ and cCompI and a2 have
    ine P (sE secret) ∨ ine Q (sE secret)
     by (simp add: TBtheorem1a)
   from this and a2 show ?thesis by auto
  next
   assume a22:¬ ine PQ (sE secret)
   from this and knowPQ and a2 have sg5:
    sKS secret ∈ LocalSecrets PQ by auto
   show ?thesis
   proof (cases know Q m)
    assume know Q m
    from this show ?thesis by simp
   next
    assume not-knowQm:¬ know Q m
    from not-knowQm and a2 have sg6a:¬ ine Q (sE secret) by simp
    from not-knowQm and a2 have sg6b:sKS secret ∉ LocalSecrets Q by simp
    show ?thesis
    proof (cases sKS secret ∈ LocalSecrets P)
     assume sKS secret ∈ LocalSecrets P
     from this and a2 show ?thesis by simp
    next
     assume sKS secret ∉ LocalSecrets P
     from sg5 and subPQ and cCompLoc and sg6a and this and sg6b have
      ine P (sE secret)
       by (simp add: LocalSecretsComposition-ine1-s)
     from this and a2 show ?thesis by simp
    qed
   qed
  qed
qed

lemma eout-knows-nonKS-k:
 assumes m ∉ (specKeys A)
        and eout A (kE m)
        and eoutKnowsECorrect A (kE m)
```

shows *knows A* [*kE m*]
using *assms*
by (*metis Expression.distinct*(*1*) *Expression.inject*(*1*) *eoutKnowsECorrect-L1*)

lemma *eout-knows-nonKS-s*:
assumes *h1*:*m* ∉ *specSecrets A*
and *h2*:*eout A* (*sE m*)
and *h3*:*eoutKnowsECorrect A* (*sE m*)
shows *knows A* [*sE m*]
using *assms*
by (*metis Expression.distinct*(*1*) *Expression.inject*(*2*) *eoutKnowsECorrect-def*)

lemma *not-knows-k-not-ine*:
assumes ¬ *knows A* [*kE m*]
shows ¬ *ine A* (*kE m*)
using *assms* **by** (*metis knows2know-neg-k not-know-k-not-ine*)

lemma *not-knows-s-not-ine*:
assumes ¬ *knows A* [*sE m*]
shows ¬ *ine A* (*sE m*)
using *assms* **by** (*metis knows2know-neg-s not-know-s-not-ine*)

lemma *not-knows-k-not-eout*:
assumes *m* ∉ *specKeys A*
and ¬ *knows A* [*kE m*]
and *eoutKnowsECorrect A* (*kE m*)
shows ¬ *eout A* (*kE m*)
using *assms* **by** (*metis eout-knows-nonKS-k*)

lemma *not-knows-s-not-eout*:
assumes *m* ∉ *specSecrets A*
and ¬ *knows A* [*sE m*]
and *eoutKnowsECorrect A* (*sE m*)
shows ¬ *eout A* (*sE m*)
using *assms* **by** (*metis eout-knows-nonKS-s*)

lemma *adv-not-knows1*:
assumes *out P* ⊆ *ins A*
and ¬ *knows A* [*kE m*]
shows ¬ *eout P* (*kE m*)
using *assms* **by** (*metis adv-not-know1 knows2know-neg-k*)

lemma *adv-not-knows2*:
assumes *out P* ⊆ *ins A*
and ¬ *knows A* [*sE m*]
shows ¬ *eout P* (*sE m*)
using *assms* **by** (*metis adv-not-know2 knows2know-neg-s*)

lemma *knows-decomposition-1-k*:

assumes *kKS a* $\notin$ *specKeysSecrets P*
and *kKS a* $\notin$ *specKeysSecrets Q*
and *subcomponents PQ* = {*P*, *Q*}
and *knows PQ* [*kE a*]
and *correctCompositionIn PQ*
and *correctCompositionLoc PQ*
shows *knows P* [*kE a*] $\vee$ *knows Q* [*kE a*]
using *assms* **by** (*metis know-decomposition knows1k*)

lemma *knows-decomposition-1-s*:
assumes *sKS a* $\notin$ *specKeysSecrets P*
and *sKS a* $\notin$ *specKeysSecrets Q*
and *subcomponents PQ* = {*P*, *Q*}
and *knows PQ* [*sE a*]
and *correctCompositionIn PQ*
and *correctCompositionLoc PQ*
shows *knows P* [*sE a*] $\vee$ *knows Q* [*sE a*]
using *assms* **by** (*metis know-decomposition knows1s*)

lemma *knows-decomposition-1*:
assumes *subcomponents PQ* = {*P*, *Q*}
and *knows PQ* [*a*]
and *correctCompositionIn PQ*
and *correctCompositionLoc PQ*
and ($\exists$ *z*. *a* = *kE z*) $\vee$ ($\exists$ *z*. *a* = *sE z*)
and $\forall$ *z*. *a* = *kE z* $\longrightarrow$
kKS z $\notin$ *specKeysSecrets P* $\wedge$ *kKS z* $\notin$ *specKeysSecrets Q*
and *h7*:$\forall$ *z*. *a* = *sE z* $\longrightarrow$
sKS z $\notin$ *specKeysSecrets P* $\wedge$ *sKS z* $\notin$ *specKeysSecrets Q*
shows *knows P* [*a*] $\vee$ *knows Q* [*a*]
using *assms*
by (*metis knows-decomposition-1-k knows-decomposition-1-s*)

lemma *knows-composition1-k*:
assumes (*kKS m*) $\notin$ *specKeysSecrets P*
and (*kKS m*) $\notin$ *specKeysSecrets Q*
and *knows P* [*kE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *knows PQ* [*kE m*]
using *assms* **by** (*metis know-composition knows1k*)

lemma *knows-composition1-s*:
assumes (*sKS m*) $\notin$ *specKeysSecrets P*
and (*sKS m*) $\notin$ *specKeysSecrets Q*
and *knows P* [*sE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*

and *correctCompositionKS PQ*
shows *knows PQ* [*sE m*]
using *assms* **by** (*metis know-composition knows1s*)

lemma *knows-composition2-k*:
assumes (*kKS m*) ∉ *specKeysSecrets P*
and (*kKS m*) ∉ *specKeysSecrets Q*
and *knows Q* [*kE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *knows PQ* [*kE m*]
using *assms*
by (*metis know2knowsPQ-k know-composition knows2know-k*)

lemma *knows-composition2-s*:
assumes (*sKS m*) ∉ *specKeysSecrets P*
and (*sKS m*) ∉ *specKeysSecrets Q*
and *knows Q* [*sE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *knows PQ* [*sE m*]
using *assms*
by (*metis know2knowsPQ-s know-composition knows2know-s*)

lemma *knows-composition-neg1-k*:
assumes *kKS m* ∉ *specKeysSecrets P*
and *kKS m* ∉ *specKeysSecrets Q*
and ¬ *knows P* [*kE m*]
and ¬ *knows Q* [*kE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionLoc PQ*
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows ¬ *knows PQ* [*kE m*]
using *assms* **by** (*metis know-decomposition knows1k*)

lemma *knows-composition-neg1-s*:
assumes *sKS m* ∉ *specKeysSecrets P*
and *sKS m* ∉ *specKeysSecrets Q*
and ¬ *knows P* [*sE m*]
and ¬ *knows Q* [*sE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionLoc PQ*
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows ¬ *knows PQ* [*sE m*]
using *assms* **by** (*metis knows-decomposition-1-s*)

lemma *knows-concat-1*:
assumes *knows P* (*a # e*)
shows *knows P* [*a*]
using *assms* **by** (*metis append-Cons append-Nil knows2*)

lemma *knows-concat-2*:
assumes *knows P* (*a # e*)
shows *knows P e*
using *assms* **by** (*metis append-Cons append-Nil knows2a*)

lemma *knows-concat-3*:
assumes *knows P* [*a*]
and *knows P e*
shows *knows P* (*a # e*)
using *assms* **by** (*metis append-Cons append-Nil knows3*)

lemma *not-knows-conc-knows-elem-not-knows-tail*:
assumes ¬ *knows P* (*a # e*)
and *knows P* [*a*]
shows ¬ *knows P e*
using *assms* **by** (*metis knows-concat-3*)

lemma *not-knows-conc-not-knows-elem-tail*:
assumes ¬ *knows P* (*a#e*)
shows ¬ *knows P* [*a*] ∨ ¬ *knows P e*
using *assms* **by** (*metis append-Cons append-Nil knows3*)

lemma *not-knows-elem-not-knows-conc*:
assumes ¬ *knows P* [*a*]
shows ¬ *knows P* (*a # e*)
using *assms* **by** (*metis knows-concat-1*)

lemma *not-knows-tail-not-knows-conc*:
assumes ¬ *knows P e*
shows ¬ *knows P* (*a # e*)
using *assms* **by** (*metis knows-concat-2*)

lemma *knows-composition3*:
fixes *e*::*Expression list*
assumes *knows P e*
and *subPQ*:*subcomponents PQ* = {*P*,*Q*}
and *cCompI*:*correctCompositionIn PQ*
and *cCompKS*:*correctCompositionKS PQ*
and ∀ (*m*::*Expression*). ((*m mem e*) ⟶
((∃ *z1*. *m* = (*kE z1*)) ∨ (∃ *z2*. *m* = (*sE z2*))))
and *notSpecKeysSecretsExpr P e*
and *notSpecKeysSecretsExpr Q e*
shows *knows PQ e*

```
using assms
proof (induct e)
  case Nil
  from this show ?case by (simp only: knows-emptyexpression)
next
  fix a l
  case (Cons a l)
  from Cons have sg1:knows P [a] by (simp add: knows-concat-1)
  from Cons have sg2:knows P l by (simp only: knows-concat-2)
  from sg1 have sg3:a mem (a # l) by simp
  from Cons and sg2 have sg2a:knows PQ l
    by (simp add: notSpecKeysSecretsExpr-L2)
  from Cons and sg1 and sg2 and sg3 show ?case
  proof (cases ∃ z1. a = kE z1)
    assume ∃ z1. a = (kE z1)
    from this obtain z where a1:a = (kE z) by auto
    from a1 and Cons have sg4:(kKS z) ∉ specKeysSecrets P
      by (simp add: notSpecKeysSecretsExpr-def)
    from a1 and Cons have sg5:(kKS z) ∉ specKeysSecrets Q
      by (simp add: notSpecKeysSecretsExpr-def)
    from sg1 and a1 have sg6:knows P [kE z] by simp
    from sg4 and sg5 and sg6 and subPQ and cCompI and cCompKS
      have knows PQ [kE z]
      by (rule knows-composition1-k)
    from this and sg2a and a1 show ?case by (simp add: knows-concat-3)
  next
    assume ¬ (∃ z1. a = kE z1)
    from this and Cons and sg3 have ∃ z2. a = (sE z2) by auto
    from this obtain z where a2:a = (sE z) by auto
    from a2 and Cons have sg8:(sKS z) ∉ specKeysSecrets P
      by (simp add: notSpecKeysSecretsExpr-def)
    from a2 and Cons have sg9:(sKS z) ∉ specKeysSecrets Q
      by (simp add: notSpecKeysSecretsExpr-def)
    from sg1 and a2 have sg10:knows P [sE z] by simp
    from sg8 and sg9 and sg10 and subPQ and cCompI and cCompKS
      have knows PQ [sE z]
      by (rule knows-composition1-s)
    from this and sg2a and a2 show ?case by (simp add: knows-concat-3)
  qed
qed

lemma knows-composition4:
 assumes knows Q e
     and subPQ:subcomponents PQ = {P,Q}
     and cCompI:correctCompositionIn PQ
     and cCompKS:correctCompositionKS PQ
     and ∀ m. m mem e ⟶ ((∃ z. m = kE z) ∨ (∃ z. m = sE z))
     and notSpecKeysSecretsExpr P e
     and notSpecKeysSecretsExpr Q e
```

shows *knows PQ e*
using *assms*
proof (*induct e*)
case *Nil*
from *this* **show** *?case* **by** (*simp only*: *knows-emptyexpression*)
next
fix *a l*
case (*Cons a l*)
from *Cons* **have** *sg1*:*knows Q* [*a*] **by** (*simp add*: *knows-concat-1*)
from *Cons* **have** *sg2*:*knows Q l* **by** (*simp only*: *knows-concat-2*)
from *sg1* **have** *sg3*:*a mem* (*a* # *l*) **by** *simp*
from *Cons* **and** *sg2* **have** *sg2a*:*knows PQ l*
by (*simp add*: *notSpecKeysSecretsExpr-L2*)
from *Cons* **and** *sg1* **and** *sg2* **and** *sg3* **show** *?case*
proof (*cases* ∃ *z1*. *a* = *kE z1*)
assume ∃ *z1*. *a* = (*kE z1*)
from *this* **obtain** *z* **where** *a1*:*a* = (*kE z*) **by** *auto*
from *a1* **and** *Cons* **have** *sg4*:(*kKS z*) ∉ *specKeysSecrets P*
by (*simp add*: *notSpecKeysSecretsExpr-def*)
from *a1* **and** *Cons* **have** *sg5*:(*kKS z*) ∉ *specKeysSecrets Q*
by (*simp add*: *notSpecKeysSecretsExpr-def*)
from *sg1* **and** *a1* **have** *sg6*:*knows Q* [*kE z*] **by** *simp*
from *sg4* **and** *sg5* **and** *sg6* **and** *subPQ* **and** *cCompI* **and** *cCompKS*
have *knows PQ* [*kE z*]
by (*rule knows-composition2-k*)
from *this* **and** *sg2a* **and** *a1* **show** *?case* **by** (*simp add*: *knows-concat-3*)
next
assume ¬ (∃ *z1*. *a* = *kE z1*)
from *this* **and** *Cons* **and** *sg3* **have** ∃ *z2*. *a* = (*sE z2*) **by** *auto*
from *this* **obtain** *z* **where** *a2*:*a* = (*sE z*) **by** *auto*
from *a2* **and** *Cons* **have** *sg8*:(*sKS z*) ∉ *specKeysSecrets P*
by (*simp add*: *notSpecKeysSecretsExpr-def*)
from *a2* **and** *Cons* **have** *sg9*:(*sKS z*) ∉ *specKeysSecrets Q*
by (*simp add*: *notSpecKeysSecretsExpr-def*)
from *sg1* **and** *a2* **have** *sg10*:*knows Q* [*sE z*] **by** *simp*
from *sg8* **and** *sg9* **and** *sg10* **and** *subPQ* **and** *cCompI* **and** *cCompKS*
have *knows PQ* [*sE z*]
by (*rule knows-composition2-s*)
from *this* **and** *sg2a* **and** *a2* **show** *?case* **by** (*simp add*: *knows-concat-3*)
qed
qed

lemma *knows-composition5*:
assumes *knows P e* ∨ *knows Q e*
and *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
and ∀ *m*. *m mem e* ⟶ ((∃ *z*. *m* = *kE z*) ∨ (∃ *z*. *m* = *sE z*))
and *notSpecKeysSecretsExpr P e*

and *notSpecKeysSecretsExpr Q e*
shows *knows PQ e*
using *assms*
by (*metis knows-composition3 knows-composition4*)

end

References

[1] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. LNCS. Springer, 2013.

[2] M. Spichkova. Stream Processing Components: Isabelle/HOL Formalisation and Case Studies. *Archive of Formal Proofs*, Nov. 2013.

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