Compositional properties of crypto-based components

Maria Spichkova

May 28, 2015

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 identifiers 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
— sets of input channels within a composed component

**definition**

- `correctCompositionIn :: specID ⇒ bool`

**where**

- `correctCompositionIn 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**

- `correctInOutLoc :: specID ⇒ bool`

**where**

- `correctInOutLoc x ≡`
  
  - `(ins x)`
  - `∩ (out x) = {}`

— Predicate insuring the correct relations between
— sets of input, output and local channels of a component

**definition**

- `correctOutLoc :: specID ⇒ bool`

**where**

- `correctOutLoc x ≡`
  
  - `(ins x)`
  - `∩ (out x) = {}`

— Predicate insuring the correct relations between
— sets of input channels within a composed component
— 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 encryption and decryption keys

axiomatization
  EncDecrKeys :: Keys ⇒ Keys ⇒ bool

where
  ExtSign:
  EncDecrKeys K1 K2 ⇒ (Ext K1 (Sign K2 E)) = E and
DecrEnc:
EncrDecrKeys K1 K2 \rightarrow (Decr K2 (Enc K1 E)) = E

— Set of private keys of a component
definition
specKeys :: specID \Rightarrow Keys set
— Set of unguessable values used by a component
definition
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) \land (x \in (specKeys C)) \} \cup
\{ z . \exists s. z = (sKS s) \land (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) \in e \rightarrow (kKS x) \notin specKeysSecrets P) \land
(\forall y. (sE y) \in e \rightarrow (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 \{} \rightarrow
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 \{} \rightarrow
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 \equiv
\text{subcomponents} \ x \neq \{\} \rightarrow
\text{specKeysSecrets} \ x = \bigcup_{} (\text{specKeysSecrets} ' (\text{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 \Rightarrow \ bool
where
correctComponentSecrecy \ x \equiv
correctCompositionKS \ x \land
\text{correctCompositionSecrets} \ x \land
\text{correctCompositionKeys} \ x \land
\text{correctCompositionLoc} \ x \land
\text{correctCompositionIn} \ x \land
\text{correctCompositionOut} \ x \land
\text{correctInOutLoc} \ x

— Predicate expr\Channel\ \text{I} \ E \text{ defines whether the expression item } E \text{ can be sent
via the channel } \text{I}
consts
eexpr\Channel :: \ chanID \Rightarrow \ Expression \Rightarrow \ bool

— Predicate eout\ M \ sP \ M \ E \text{ defines whether the component } sP \text{ may eventually
— output an expression } E \text{ if there exists a time interval } t \text{ of
— an output channel which contains this expression } E
definition
eout :: \ specID \Rightarrow \ Expression \Rightarrow \ bool
where
eout \ sP \ E \equiv
\exists \ (ch :: chanID). ((ch \in (\text{out} \ sP)) \land (eexpr\Channel \ ch \ E))

— Predicate eout \ sP \ E \text{ defines whether the component } sP \text{ may eventually
— output an expression } E \text{ via subset of channels } M,
— which is a subset of output channels of } sP,
— and if there exists a time interval } t \text{ of
— an output channel which contains this expression } E
definition
eoutM :: \ specID \Rightarrow chanID \ set \Rightarrow \ Expression \Rightarrow \ bool
where
eoutM \ sP \ M \ E \equiv
\exists \ (ch :: chanID). ((ch \in (\text{out} \ sP)) \land (ch \in M) \land (eexpr\Channel \ ch \ E))

— Predicate ine\ M \ sP \ M \ E \text{ defines whether a component } sP \text{ may eventually
— get an expression } E \text{ if there exists a time interval } t \text{ of
— an input stream which contains this expression } E
definition
ine :: \ specID \Rightarrow \ Expression \Rightarrow \ bool
where

\[
\text{ine } sP \ E \equiv
\exists \ (ch :: \text{chanID}). ((ch \in (\text{ins } sP)) \land (\text{exprChannel } ch \ E))
\]

— Predicate \(\text{ine } sP \ E\) 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**

\(\text{ineM } :: \text{specID} \Rightarrow \text{chanID set} \Rightarrow \text{Expression} \Rightarrow \text{bool}\)

where

\[
\text{ineM } sP \ M \ E \equiv
\exists \ (ch :: \text{chanID}). ((ch \in (\text{ins } sP)) \land (ch \in M) \land (\text{exprChannel } ch \ E))
\]

— 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**

\(\text{out-exprChannelSingle } :: \text{specID} \Rightarrow \text{chanID} \Rightarrow \text{Expression} \Rightarrow \text{bool}\)

where

\[
\text{out-exprChannelSingle } sP \ ch \ E \equiv
(ch \in (\text{out } sP)) \land
(\text{exprChannel } ch \ E) \land
(\forall (x :: \text{chanID}) \ (t :: \text{nat}). ((x \in (\text{out } sP)) \land (x \neq ch) \rightarrow \neg \text{exprChannel } x \ E))
\]

— This predicate yields true if only the channels from the set \(\text{chSet}\),
— which is a subset of input channels of the component \(sP\),
— may eventually output an expression \(E\)

**definition**

\(\text{out-exprChannelSet } :: \text{specID} \Rightarrow \text{chanID set} \Rightarrow \text{Expression} \Rightarrow \text{bool}\)

where

\[
\text{out-exprChannelSet } sP \ chSet \ E \equiv
((\forall (x :: \text{chanID}) . ((x \in \text{chSet}) \rightarrow (x \in (\text{out } sP)) \land (\text{exprChannel } x \ E))))
\land
((\forall (x :: \text{chanID}) . ((x \notin \text{chSet}) \land (x \in (\text{out } sP)) \rightarrow \neg \text{exprChannel } x \ E)))
\]

— 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 get an expression \(E\)

**definition**

\(\text{ine-exprChannelSingle } :: \text{specID} \Rightarrow \text{chanID} \Rightarrow \text{Expression} \Rightarrow \text{bool}\)

where

\[
\text{ine-exprChannelSingle } sP \ ch \ E \equiv
(ch \in (\text{ins } sP)) \land
(\text{exprChannel } ch \ E) \land
(\forall (x :: \text{chanID}) \ (t :: \text{nat}). ((x \in (\text{ins } sP)) \land (x \neq ch) \rightarrow \neg \text{exprChannel } x \ E))
\]

7
— 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**

\[
in\text{-exprChannelSet} :: \text{specID} \Rightarrow \text{chanID set} \Rightarrow \text{Expression} \Rightarrow \text{bool}
\]

**where**

\[
in\text{-exprChannelSet} \ sP \ chSet \ E \equiv
\]

\[
(\forall (x :: \text{chanID}). ((x \in \text{chSet}) \rightarrow ((x \in \text{ins sP}) \land (\text{exprChannel} \ x \ E))))
\]

\[
\land
\]

\[
(\forall (x :: \text{chanID}). ((x \not\in \text{chSet}) \land (x \in \text{ins sP}) \rightarrow \neg \text{exprChannel} \ x \ E))
\]

— 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 composition 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 $ch \not\in \text{loc PQ}$

and $ch \in \text{ins P}$

**shows** $ch \in \text{ins PQ}$

**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 $ch \in \text{ins PQ}$

**shows** $(ch \in \text{ins P}) \lor (ch \in \text{ins Q})$

**using** assms by (simp add: correctCompositionIn-def)

**lemma** ineM-L1:

**assumes** $ch \in M$
and \( ch \in \text{ins} P \)
and \( \text{exprChannel} ch E \)
shows \( \text{ineM} P M E \)
using assms by (simp add: ineM-def, blast)

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

lemma not-ine-ineM:
assumes \( \neg \text{ine} P E \)
shows \( \neg \text{ineM} P M E \)
using assms by (simp add: ineM-def ine-def)

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

lemma not-eout-eoutM:
assumes \( \neg \text{eout} P E \)
shows \( \neg \text{eoutM} P M E \)
using assms by (simp add: eoutM-def eout-def)

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

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

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

lemma correctCompositionSecrets-subcomp2:
assumes \( \text{correctCompositionSecrets} C \)
and \( xb \in \text{subcomponents} C \)
and \( x_C \in \text{specSecrets} x_B \)
shows \( x_C \in \text{specSecrets} C \)
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 \( C \neq \{\} \)
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 \( C \neq \{\} \)
from this and assms show \(?thesis\)
by (simp add: correctCompositionKS-def
correctCompositionSecrets-def
specKeysSecrets-def, blast)

qed

**Lemma correctCompKS-KeysSecrets:**
**Assumes** correctCompositionSecrets \( C \)
and correctCompositionSecrets \( C \)
**Shows** correctCompositionKS \( C \)
**Proof** (cases subcomponents \( C = \{\} \))
assume subcomponents \( C = \{\} \)
from this and assms show \(?thesis\)
by (simp add: correctCompositionKS-def
correctCompositionKeys-def
correctCompositionSecrets-def
specKeysSecrets-def, blast)
lemma correctCompositionKS-subcomp1:
  assumes correctCompositionKS C and h1: x ∈ subcomponents C and xa ∈ specKeys C
  shows \( \exists y \in \text{subcomponents } C. \ (xa \in \text{specKeys } y) \)
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 ∈ subcomponents C and xa ∈ specSecrets C
  shows \( \exists y \in \text{subcomponents } C. \ xa \in \text{specSecrets } y \)
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 ∈ subcomponents C and xa ∈ specKeys x
  shows xa ∈ specKeys C
using assms
by (simp add: correctCompositionKS-def specKeysSecrets-def, auto)

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

lemma correctCompositionKS-PQ:
  assumes subcomponents PQ = \{P, Q\} and correctCompositionKS PQ
and \( ks \in \text{specKeysSecrets PQ} \)
shows \( ks \in \text{specKeysSecrets P} \lor ks \in \text{specKeysSecrets Q} \)
using assms by (simp add: correctCompositionKS-def)

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

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

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

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

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

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

lemma ine-exprChannelSet-Single:
assumes ine-exprChannelSet P \{ ch \} E
shows ine-exprChannelSingle P ch E
using assms by (simp add: ine-exprChannelSingle-def ine-exprChannelSet-def)
lemma ine-ins-neg1:
assumes \( \neg \text{ine} P \ m \)
and \( \text{exprChannel} \ x \ m \)
shows \( x \notin \text{ins} P \)
using assms by (simp add: ine-def, auto)

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

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

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

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

lemma correctCompositionIn-prop1:
assumes \( \text{subcomponents} \ PQ = \{P, Q\} \)
and \( \text{correctCompositionIn} PQ \)
and \( x \in (\text{ins} PQ) \)
shows \( (x \in (\text{ins} P)) \lor (x \in (\text{ins} Q)) \)
using assms by (simp add: correctCompositionIn-def)

lemma correctCompositionOut-prop1:
assumes \( \text{subcomponents} \ PQ = \{P, Q\} \)
and \( \text{correctCompositionOut} PQ \)
and \( x \in (\text{out} PQ) \)
shows \( (x \in (\text{out} P)) \lor (x \in (\text{out} Q)) \)
using assms by (simp add: correctCompositionOut-def)
theorem TBtheorem3a:
assumes \( \neg (\text{ine } P \ E) \) and \( \neg (\text{ine } Q \ E) \)
and subcomponents \( PQ = \{P,Q\}\)
and correctCompositionIn \( PQ \)
shows \( \neg (\text{ine } PQ \ E) \)
using assms by (simp add: ine-def correctCompositionIn-def, auto)

theorem TBlemma3b:
assumes \( h1: \neg (\text{ineM } P \ M \ E) \)
and \( h2: \neg (\text{ineM } Q \ M \ E) \)
and subPQ: \( \text{subcomponents } PQ = \{P,Q\}\)
and cCompI: correctCompositionIn \( PQ \)
and chM: \( \text{ch} \in M \)
and chPQ: \( \text{ch} \in \text{ins } PQ \)
and eCh: exprChannel \( \text{ch} \ E \)
shows False
proof (cases \( \text{ch} \in \text{ins } P \))
  assume a1: \( \text{ch} \in \text{ins } P \)
  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: \( \text{ch} \notin \text{ins } P \)
  from subPQ and cCompI and chPQ have \( (\text{ch} \in \text{ins } P) \lor (\text{ch} \in \text{ins } Q) \)
    by (simp add: correctCompositionIn-L2)
  from this and a2 have \( \text{ch} \in \text{ins } Q \) 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 \( \neg (\text{ineM } P \ M \ E) \)
and \( \neg (\text{ineM } Q \ M \ E) \)
and subcomponents \( PQ = \{P,Q\}\)
and correctCompositionIn \( PQ \)
shows \( \neg (\text{ineM } PQ \ M \ E) \)
using assms by (metis TBtheorem1b)

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

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

theorem \( TB\text{theorem4b-P} \):
assumes ineM P M E
and subcomponents PQ = \{ P, Q \}
and correctCompositionIn PQ
and \( \exists \) ch. \(((ch \in (\text{ins} P)) \lor (ch \in (\text{ins} Q)) \land \text{exprChannel} ch E \land \neg (ch \in (\text{loc PQ})))\)
shows ineM PQ M E
using assms by (simp add: ineM-def correctCompositionIn-def, auto)

theorem \( TB\text{theorem4a-PQ} \):
assumes (ine P E) \lor (ine Q E)
and subcomponents PQ = \{ P, Q \}
and correctCompositionIn PQ
and \( \exists \) ch. \(((ch \in (\text{ins} P)) \lor (ch \in (\text{ins} Q)) \land \text{exprChannel} ch E \land \neg (ch \in (\text{loc PQ})))\)
shows ine PQ E
using assms by (simp add: ine-def correctCompositionIn-def, auto)

theorem \( TB\text{theorem4b-PQ} \):
assumes (ineM P M E) \lor (ineM Q M E)
and subcomponents PQ = \{ P, Q \}
and correctCompositionIn PQ
and \( \exists \) ch. \(((ch \in (\text{ins} P)) \lor (ch \in (\text{ins} Q)) \land \text{exprChannel} ch E \land \neg (ch \in (\text{loc PQ})))\)
shows ineM PQ M E
using assms by (simp add: ineM-def correctCompositionIn-def, auto)

theorem \( TB\text{theorem4a-notP1} \):
assumes ine P E
and \neg ine Q E
and subcomponents PQ = \{ P, Q \}
and correctCompositionIn PQ
and \( \exists \) ch. \(((ine-exprChannelSingle P ch E) \land (ch \in (\text{loc PQ})))\)
shows \neg ine PQ E
using assms by (simp add: ine-def correctCompositionIn-def ine-exprChannelSingle-def, auto)

theorem \( TB\text{theorem4b-notP1} \):
assumes ineM P M E
and \neg ineM Q M E
and subcomponents PQ = \{ P, Q \}
and correctCompositionIn PQ
and \( \exists \ ch. \ ((\text{ine-exprChannelSingle} P \ ch \ E) \land (ch \in M) \land (ch \in (\text{loc} PQ))) \)

shows \( \neg \text{ineM PQ M E} \)

using assms
by (simp add: ineM-def correctCompositionIn-def
ine-exprChannelSingle-def, auto)

theorem TBtheorem4a-notP2:
assumes \( \neg \text{ine} \ Q \ E \)
and subcomponents \( PQ = \{P,Q\} \)
and correctCompositionIn \( PQ \)
and \( \text{ine-exprChannelSet} P \ ChSet E \)
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSet}) \rightarrow (x \in (\text{loc} PQ))) \)

shows \( \neg \text{ine} \ PQ E \)
using assms
by (simp add: ine-def correctCompositionIn-def
ine-exprChannelSet-def, auto)

theorem TBtheorem4b-notPQ:
assumes \( \neg \text{ineM} \ Q \ M \ E \)
and subcomponents \( PQ = \{P,Q\} \)
and correctCompositionIn \( PQ \)
and \( \text{ine-exprChannelSet} P \ ChSet E \)
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSet}) \rightarrow (x \in (\text{loc} PQ))) \)

shows \( \neg \text{ineM PQ M E} \)
using assms
by (simp add: ineM-def correctCompositionIn-def
ine-exprChannelSet-def, auto)

lemma ineM-Un1:
assumes \( \text{ineM} \ P \ A \ E \)

shows \( \text{ineM} \ P \ (A \ Un \ B) \ E \)
using assms by (simp add: ineM-def, auto)

theorem TBtheorem4b-notPQ:
assumes subcomponents \( PQ = \{P,Q\} \)
and correctCompositionIn \( PQ \)

and \( \forall (x :: \text{chanID}). ((x \in \text{ChSetP}) \rightarrow (x \in (\text{loc} PQ))) \)
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSetQ}) \rightarrow (x \in (\text{loc} PQ))) \)

shows \( \neg \text{ine PQ E} \)
using assms
by (simp add: ineM-def correctCompositionIn-def
ine-exprChannelSetDef, auto)
and \( \text{in-exprChannelSet} \, P \, \text{ChSet} \, P \, E \)
and \( \text{in-exprChannelSet} \, Q \, \text{ChSet} \, Q \, E \)
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSet}P) \rightarrow (x \in (\text{loc} \, PQ))) \)
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSet}Q) \rightarrow (x \in (\text{loc} \, PQ))) \)
shows \( \neg \text{ineM} \, PQ \, M \, E \)
using assms
by (simp add: inM-def correctCompositionIn-def
\text{in-exprChannelSet-def}, auto)

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

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

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

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

**theorem** TBtheorem54b-P:
assumes \( \text{eout} \, M \, P \, M \, E \)
and subcomponents \( PQ = \{P, Q\} \)
and correctCompositionOut \( PQ \)
and \( \exists \text{ch}. ((\text{ch} \in (\text{out} \, Q)) \land (\text{exprChannel} \, \text{ch} \, E) \land \text{ch} \notin (\text{loc} \, PQ)) \land (\text{ch} \in M) \) )
shows \( \text{eout} \, M \, PQ \, M \, E \)
using assms by (simp add: eoutM-def correctCompositionOut-def, auto)

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

theorem TBtheorem5b-PQ:
assumes (eoutM P M E) \lor (eoutM Q M E)
and subcomponents PQ = \{P, Q\}
and correctCompositionOut PQ
and \(\exists ch. (((ch \in (out P)) \lor (ch \in out Q)) \land (ch \in M) \land (\text{exprChannel ch } E) \land (ch \notin \text{loc } PQ)))\)
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 \(\exists ch. ((\text{out-exprChannelSingle } P ch \ E) \land (ch \in \text{loc } PQ)))\)
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 \(\exists ch. ((\text{out-exprChannelSingle } P ch \ E) \land (ch \in M) \land (ch \notin \text{loc } PQ)))\)
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 \(\forall x :: \text{chanID}. ((x \in \text{ChSet}) \longrightarrow (x \in \text{loc } PQ)))\)
shows \(\neg eout PQ E\)
using assms
by (simp add: eout-def correctCompositionOut-def
out-exprChannelSet-def, auto)
theorem TBtheorem5b-notP2:
assumes \( \neg e_{\text{out}\text{M}} Q M E \)
and subcomponents \( PQ = \{P,Q\} \)
and correctCompositionOut PQ
and out-exprChannelSet P ChSet E
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSet}) \longrightarrow (x \in (\text{loc} PQ))) \)
shows \( \neg e_{\text{out}\text{M}} PQ M E \)
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 ChSet P E
and out-exprChannelSet Q ChSet Q E
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSetP}) \longrightarrow (x \in (\text{loc} PQ))) \)
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSetQ}) \longrightarrow (x \in (\text{loc} PQ))) \)
shows \( \neg e_{\text{out}} PQ E \)
using assms
by (simp add: eout-def correctCompositionOut-def out-exprChannelSet-def, auto)

/* Also states that the local secrets are not transmitted to the external environment. */

theorem TBtheorem5b-notPQ:
assumes subcomponents \( PQ = \{P,Q\} \)
and correctCompositionOut PQ
and out-exprChannelSet P ChSet P E
and out-exprChannelSet Q ChSet Q E
and \( M = \text{ChSetP} \cup \text{ChSetQ} \)
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSetP}) \longrightarrow (x \in (\text{loc} PQ))) \)
and \( \forall (x :: \text{chanID}). ((x \in \text{ChSetQ}) \longrightarrow (x \in (\text{loc} PQ))) \)
shows \( \neg e_{\text{out}\text{M}} PQ M E \)
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

end
LocalSecrets :: specID ⇒ KS set

where
LocalSecretsDef:
LocalSecrets A = 
\{(m :: KS). m ∉ specKeysSecrets A \land
\((\exists x y. ((x ∈ loc A) \land m = (kKS y) \land (exprChannel x (kE y))))\)
\union (\exists x z. ((x ∈ loc A) \land m = (sKS z) \land (exprChannel x (sE z)) ))})\}

lemma LocalSecretsComposition1:
assumes ls ∈ LocalSecrets P
and subcomponents PQ = \{P, Q\}
shows ls ∈ LocalSecrets PQ
using assms by (simp (no-asn) 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 \land 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 \land 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-asn) 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 \( \text{correctCompositionKS PQ} \)
and \( (kKS m) \not\in \text{specKeysSecrets P} \)
and \( (kKS m) \not\in \text{specKeysSecrets Q} \)
and \( \neg \text{ine P} (kE m) \)
and \( \neg \text{ine Q} (kE m) \)
and \( (kKS m) \not\in ((\text{LocalSecrets P}) \cup (\text{LocalSecrets Q})) \)
shows \( (kKS m) \not\in (\text{LocalSecrets PQ}) \)

proof –
from \text{assms show } \neg \text{thesis}
apply (simp (no-asms) only: \text{LocalSecretsDef},
simp add: \text{correctCompositionLoc-def, clarify})
by (rule \text{LocalSecretsComposition-exprChannel-k, auto})
qed

lemma \text{LocalSecretsComposition-neg-s}:
assumes \( \text{subcomponents PQ} = \{P, Q\} \)
and \( \text{cCompLoc}: \text{correctCompositionLoc PQ} \)
and \( \text{cCompKS}: \text{correctCompositionKS PQ} \)
and \( \text{notKSP}: (sKS m) \not\in \text{specKeysSecrets P} \)
and \( \text{notKSQ}: (sKS m) \not\in \text{specKeysSecrets Q} \)
and \( \neg \text{ine P} (sE m) \)
and \( \neg \text{ine Q} (sE m) \)
and \( \neg \text{LocSeqPQ}: (sKS m) \not\in ((\text{LocalSecrets P}) \cup (\text{LocalSecrets Q})) \)
shows \( (sKS m) \not\in (\text{LocalSecrets PQ}) \)

proof –
from \text{subPQ and cCompKS and notKSP and notKSQ}
have \( \text{sg1}: sKS m \not\in \text{specKeysSecrets PQ} \)
by (simp add: \text{correctCompositionKS-neg1})
from \text{subPQ and cCompLoc and notLocSeqPQ have sg2:}
\( sKS m \not\in (\text{LocalSecrets} \cdot \text{subcomponents PQ}) \)
by simp
from \text{sg1 and sg2 and assms show } \neg \text{thesis}
apply (simp (no-asms) only: \text{LocalSecretsDef},
simp add: \text{correctCompositionLoc-def, clarify})
by (rule \text{LocalSecretsComposition-exprChannel-s, auto})
qed

lemma \text{LocalSecretsComposition-neg}:
assumes \( \text{subcomponents PQ} = \{P, Q\} \)
and \( \text{correctCompositionLoc PQ} \)
and \( \text{correctCompositionKS PQ} \)
and \( ks \notin \text{specKeysSecrets P} \)
and \( ks \notin \text{specKeysSecrets Q} \)
and \( h1:\forall m. \ ks = kKS m \rightarrow (\neg \text{ine P} (kE m) \land \neg \text{ine Q} (kE m)) \)
and \( h2:\forall m. \ ks = sKS m \rightarrow (\neg \text{ine P} (sE m) \land \neg \text{ine Q} (sE m)) \)
and \( ks \notin ((\text{LocalSecrets P}) \cup (\text{LocalSecrets Q})) \)
shows \( ks \notin (\text{LocalSecrets PQ}) \)
proof (cases ks)
fix m
assume \( a_1 \): \( k_\text{S} = k\text{K}_\text{S} m \)
from this and \( h_1 \) have  \( \neg \text{ine} P \ (k E \ m) \land \neg \text{ine} Q \ (k E \ m) \) by simp
from this and \( a_1 \) and assms show \( ?\text{thesis} \)
  by (simp add: \text{LocalSecretsComposition-neg-k})

next
fix \( m \)
assume \( a_2 \): \( k_\text{S} = s\text{K}_\text{S} m \)
from this and \( h_2 \) have  \( \neg \text{ine} P \ (s E \ m) \land \neg \text{ine} Q \ (s E \ m) \) by simp
from this and \( a_2 \) and assms show \( ?\text{thesis} \)
  by (simp add: \text{LocalSecretsComposition-neg-s})

qed

lemma \text{LocalSecretsComposition-neg1-s}:
assumes \( \text{subcomponents PQ} = \{P, \ Q\} \)
  and \( \text{correctCompositionLoc PQ} \)
  and  \( \neg \text{ine} P \ (s E \ s) \)
  and  \( \neg \text{ine} Q \ (s E \ s) \)
  and \( k_\text{S} s \notin \text{LocalSecrets} \ P \)
  and \( k_\text{S} s \notin \text{LocalSecrets} \ Q \)
shows  \( k_\text{S} s \notin \text{LocalSecrets} \ PQ \)
proof (cases \( k_\text{S} \))
fix \( m \)
assume \( a_1 \): \( k_\text{S} = k\text{K}_\text{S} m \)
from this and \( h_1 \) have  \( \neg \text{ine} P \ (k E \ m) \land \neg \text{ine} Q \ (k E \ m) \) by simp
from this and \( a_1 \) and assms show \( ?\text{thesis} \)
  by (simp add: \text{LocalSecretsComposition-neg1-k})

next
fix \( m \)
assume \( a_2 \): \( k_\text{S} = s\text{K}_\text{S} m \)
from this and \( h_2 \) have  \( \neg \text{ine} P \ (s E \ m) \land \neg \text{ine} Q \ (s E \ m) \) by simp
from this and \( a_2 \) and assms show \( ?\text{thesis} \)

22
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-neg1-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-neg1-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 \text{LocalSecrets } P
  and \text{exprChannel } ch (sE secret)
  and sKS secret \notin \text{specKeysSecrets } P
shows ch \notin \text{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 \text{LocalSecrets } PQ
  and ch \in \text{ins } P
  and \text{exprChannel } ch (kE key)
  and kKS key \notin \text{specKeysSecrets } PQ
  and correctCompositionIn PQ
shows ch \in \text{ins } PQ \land \text{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 \text{LocalSecrets } PQ
  and ch \in \text{ins } P
  and \text{exprChannel } ch (kE key)
  and kKS key \notin \text{specKeysSecrets } PQ
  and correctCompositionIn PQ
shows \exists ch. ch \in \text{ins } PQ \land \text{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 \text{LocalSecrets } PQ
  and ch \in \text{ins } Q
  and h1: \text{exprChannel } ch (kE key)
  and kKS key \notin \text{specKeysSecrets } PQ
  and correctCompositionIn PQ
shows ch \in \text{ins } PQ \land \text{exprChannel } ch (kE key)
proof
  from assms have ch \notin \text{loc } PQ
  by (simp add: LocalSecretsComposition-neg-loc-k)
from this and assms have ch \in \text{ins } PQ
  by (simp add: correctCompositionIn-def)
from this and h1 show \theta \text{thesis} by simp
qed
Lemma \( \text{correctCompositionKS-exprChannel-k-Q} \):

**Assumes**
- \( \text{subcomponents } PQ = \{P, Q\} \)
- \( \text{correctCompositionKS } PQ \)
- \( kKS \text{ key } \notin \text{LocalSecrets } PQ \)
- \( ch \notin \text{ins } Q \)
- \( \text{exprChannel } ch \ (kE \text{ key}) \)
- \( kKS \text{ key } \notin \text{specKeysSecrets } PQ \)
- \( \text{correctCompositionIn } PQ \)

**Shows**
- \( \exists \ ch. \ ch \in \text{ins } PQ \land \text{exprChannel } ch \ (kE \text{ key}) \)

**Using** \( \text{assms} \)

**By** \((\text{metis correctCompositionKS-exprChannel-k-Q})\)

Lemma \( \text{correctCompositionKS-exprChannel-s-P} \):

**Assumes**
- \( \text{subcomponents } PQ = \{P, Q\} \)
- \( \text{correctCompositionKS } PQ \)
- \( sKS \text{ secret } \notin \text{LocalSecrets } PQ \)
- \( ch \in \text{ins } P \)
- \( \text{exprChannel } ch \ (sE \text{ secret}) \)
- \( sKS \text{ secret } \notin \text{specKeysSecrets } PQ \)
- \( \text{correctCompositionIn } PQ \)

**Shows**
- \( ch \in \text{ins } PQ \land \text{exprChannel } ch \ (sE \text{ secret}) \)

**Using** \( \text{assms} \)

**By** \((\text{metis LocalSecretsComposition-neg-loc-s correctCompositionIn-L1})\)

Lemma \( \text{correctCompositionKS-exprChannel-s-Pex} \):

**Assumes**
- \( \text{subcomponents } PQ = \{P, Q\} \)
- \( \text{correctCompositionKS } PQ \)
- \( sKS \text{ secret } \notin \text{LocalSecrets } PQ \)
- \( ch \in \text{ins } P \)
- \( \text{exprChannel } ch \ (sE \text{ secret}) \)
- \( sKS \text{ secret } \notin \text{specKeysSecrets } PQ \)
- \( \text{correctCompositionIn } PQ \)

**Shows**
- \( \exists ch. \ ch \in \text{ins } PQ \land \text{exprChannel } ch \ (sE \text{ secret}) \)

**Using** \( \text{assms} \)

**By** \((\text{metis correctCompositionKS-exprChannel-s-P})\)

Lemma \( \text{correctCompositionKS-exprChannel-s-Q} \):

**Assumes**
- \( \text{subcomponents } PQ = \{P, Q\} \)
- \( \text{correctCompositionKS } PQ \)
- \( sKS \text{ secret } \notin \text{LocalSecrets } PQ \)
- \( ch \in \text{ins } Q \)
- \( \text{h1:exprChannel } ch \ (sE \text{ secret}) \)
- \( sKS \text{ secret } \notin \text{specKeysSecrets } PQ \)
- \( \text{correctCompositionIn } PQ \)

**Shows**
- \( ch \in \text{ins } PQ \land \text{exprChannel } ch \ (sE \text{ secret}) \)

**Proof**

**From** \( \text{assms} \) \( \text{have } ch \notin \text{loc } PQ \)

**By** \((\text{simp add: LocalSecretsComposition-neg-loc-s})\)

**From** \( \text{this and assms} \) \( \text{have } ch \in \text{ins } PQ \)
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 \notin LocalSecrets PQ
  and ch \in ins Q
  and exprChannel ch (sE secret)
  and sKS secret \notin specKeysSecrets PQ
  and correctCompositionIn PQ
shows \exists ch. ch \in ins PQ \land 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 secrets, 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 concatenation 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 \Rightarrow KS \Rightarrow bool

where

know A (kKS m) =
  ((ine A (kE m)) \lor ((kKS m) \in (LocalSecrets A))) |
know A (sKS m) =
\[(\text{ine } A (sE m)) \lor ((sKS m) \in (\text{LocalSecrets } A))]\]

axiomatization

\[\text{knows} :: \text{specID } \Rightarrow \text{Expression } \Rightarrow \text{bool}\]

where

knows-emptyexpression:

\[\text{knows } C \[\] = \text{True and}\]

know1k:

\[\text{knows } C \[KS2Expression (kKS m1)\] = \text{know } C (kKS m1) \text{ and}\]

know1s:

\[\text{knows } C \[KS2Expression (sKS m2)\] = \text{know } C (sKS m2) \text{ and}\]

knows2a:

\[\text{knows } A (e1 \oplus e) \rightarrow \text{knows } A e \text{ and}\]

knows2b:

\[\text{knows } A (e \oplus e1) \rightarrow \text{knows } A e \text{ and}\]

knows3:

\[(\text{knows } A c1) \land (\text{knows } A c2) \rightarrow \text{knows } A (e1 \oplus e2) \text{ and}\]

knows4:

\[(\text{IncrDecrKeys } k1 k2) \land (\text{know } A (kKS k1)) \land (\text{knows } A (\text{Enc } k1 e)) \rightarrow \text{knows } A e\]

and

knows5:

\[(\text{IncrDecrKeys } k1 k2) \land (\text{know } A (kKS k1)) \land (\text{knows } A (\text{Sign } k2 e)) \rightarrow \text{knows } A e\]

and

knows6:

\[(\text{know } A (kKS k)) \land (\text{knows } A e1) \rightarrow \text{knows } A (\text{Enc } k e1)\]

and

knows7:

\[(\text{know } A (kKS k)) \land (\text{knows } A e1) \rightarrow \text{knows } A (\text{Sign } k e1)\]

primrec eoutKnowCorrect :: specID \Rightarrow KS \Rightarrow bool

where

eout-know-k:

\[\text{eoutKnowCorrect } C (kKS m) =\]

\[(\text{eout } C (kE m)) \iff (m \in (\text{specKeys } C) \lor (\text{know } C (kKS m))))\]


eout-know-s:

\[\text{eoutKnowCorrect } C (sKS m) =\]

\[(\text{eout } C (sE m)) \iff (m \in (\text{specSecrets } C) \lor (\text{know } C (sKS m))))\]

definition eoutKnowsECorrect :: specID \Rightarrow Expression \Rightarrow bool

where

eoutKnowsECorrect C e \equiv

\[(\text{eout } C e) \iff

(\exists k. e = (kE k) \land (k \in \text{specKeys } C)) \lor

(\exists s. e = (sE s) \land (s \in \text{specSecrets } C)) \lor

(\text{knows } C [e]))\]

lemma eoutKnowCorrect-L1k:
assumes \( \text{eoutKnowCorrect} \ C \ (kKS \ m) \)
and \( \text{eout} \ C \ (kE \ m) \)
shows \( m \in (\text{specKeys} \ C) \lor (\text{know} \ C \ (kKS \ m)) \)
using assms by (metis eout-know-k)

lemma eoutKnowCorrect-L1s:
assumes \( \text{eoutKnowCorrect} \ C \ (sKS \ m) \)
and \( \text{eout} \ C \ (sE \ m) \)
shows \( m \in (\text{specSecrets} \ C) \lor (\text{know} \ C \ (sKS \ m)) \)
using assms by (metis eout-know-s)

lemma eoutKnowsECorrect-L1:
assumes \( \text{eoutKnowsECorrect} \ C \ e \)
and \( \text{eout} \ C \ e \)
shows \( \exists k. e = (kE \ k) \land (k \in \text{specKeys} \ C) \lor \\
(\exists s. e = (sE \ s) \land (s \in \text{specSecrets} \ C) \lor \\
(\text{knows} \ C \ [e]) \)
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 \( (kKS \ m) \lor \text{know} \ Q \ (kKS \ m) \)
shows knows P \[kE \ m\] \lor knows Q \[kE \ m\]
using assms by (metis know2knows-k)

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

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

lemma know2knows-neg-k:
assumes \( \neg \text{know A} \ (kKS \ m) \)
shows \( \neg \text{knows A} \ [kE \ m] \)
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 know2know-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 know2know-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 know2know-s)

lemma knows2know-neg-s:
assumes ¬ knows A [sE m]
shows ¬ know A (sKS m)
using assms by (metis know2know-s)

lemma know2:
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 \( ch : \text{ins} P \)
and exprChannel ch m
and \( \forall x. x \in \text{ins} PQ \rightarrow \neg \text{exprChannel} x m \)
shows \( ch : \text{loc} PQ \)
using assms by (simp add: correctCompositionIn-def, auto)

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

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

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

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

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

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

lemma adv-not-know1:
assumes \( \text{out} P \subseteq \text{ins} A \)
and \( \neg \text{know} A (kKS m) \)
shows \( \neg \text{eout } P \) (\( kE \) m)
using assms
by (metis (full-types) eout-def ine-ins-neg1 not-know-k-not-ine set-rev-mp)

lemma adv-not-know2:
assumes out \( P \subseteq \text{ins } A \)
and \( \neg \text{know } A \) (\( sKS \) m)
shows \( \neg \text{eout } P \) (\( sE \) m)
using assms
by (metis (full-types) eout-def ine-ins-neg1 not-know-s-not-ine set-rev-mp)

lemma LocalSecrets-L1:
assumes \( (kKS) \) key \( \in \text{LocalSecrets } P \)
and \( (kKS \text{ key}) \notin \bigcup (\text{LocalSecrets } ^c \text{ subcomponents } P) \)
shows \( kKS \text{ key} \notin \text{specKeysSecrets } P \)
using assms by (simp only: LocalSecretsDef, auto)

lemma LocalSecrets-L2:
assumes \( kKS \text{ key} \in \text{LocalSecrets } P \)
and \( kKS \text{ key} \in \text{specKeysSecrets } P \)
shows \( kKS \text{ key} \in \bigcup (\text{LocalSecrets } ^c \text{ subcomponents } P) \)
using assms by (simp only: LocalSecretsDef, auto)

lemma know-composition1:
assumes notKSP: \( m \notin \text{specKeysSecrets } P \)
and notKSQ: \( m \notin \text{specKeysSecrets } Q \)
and know P m
and subPQ: subcomponents PQ = \{P, Q\}
and cCompI: correctCompositionIn PQ
and cCompKS: correctCompositionKS PQ
shows \( \text{know } PQ \) m
proof (cases m)
  fix key
  assume a1: \( m = kKS \) key
  show \( ? \text{thesis} \)
  proof (cases ine P (\( kE \) key))
    assume a11: ine P (\( kE \) key)
    from this have a11text: ine P (\( kE \) key) \( | \) ine Q (\( kE \) key) by simp
    from subPQ and cCompKS and notKSP and notKSQ
    have m \( \notin \) specKeysSecrets PQ
      by (rule correctCompositionKS-neg1)
    from this and a1 have sg1: kKS key \( \notin \) specKeysSecrets PQ by simp
    from a1 and a11text and cCompKS show \( ? \text{thesis} \)
    proof (cases loc PQ = {})
      assume a1locE: loc PQ = {}
      from a11text and subPQ and cCompI and a1locE have ine PQ (\( kE \) key)
        by (rule TBtheorem4a-empty)
      from this and a1 show \( ? \text{thesis} \) by auto
  next
31
assume $a_{11}ocalNE:loc PQ \neq \{\}$ from $a_1$ and $a_{11}$ and $sg_1$ and assms show $\text{thesis}$  
apply (simp add: ine-def, auto)  
by (simp add: correctCompositionKS-exprChannel-k-Pex)
qed

next
assume $a_{12}:\neg ine P (kE key)$ from this and $a_1$ and assms show $\text{thesis}$  
by (auto, simp add: LocalSecretsComposition1)
qed

next
fix secret
assume $a_{2}:m = sKS secret$ show $\text{thesis}$
  proof (cases $ine P (sE secret)$)
    assume $a_{21}:ine P (sE secret)$
    from this have $a_{21}\text{ext}:ine P (sE secret) | ine Q (sE secret)$ by simp
    from $\text{subPQ and cCompKS and notKSP and notKSQ have } m \notin \text{specKeysSecrets PQ}$
      by (rule correctCompositionKS-neg1)
    from this and $a_2$ have $\text{sg}_2:sKS secret \notin \text{specKeysSecrets PQ}$ by simp
    from $\text{a}_{2}\text{ and a}_{2}\text{ext and cCompKS show }\text{thesis}$
      assume $a_{21}\text{ext}:loc PQ = \{\}$
    from $a_{2}\text{ext and subPQ and cCompI and a}_{21}\text{locE have ine PQ (sE secret)}$
      by (rule TBtheorem4a-empty)
    from this and $a_2$ show $\text{thesis}$ by auto
    next
    assume $a_{21}\text{locNE:loc PQ \neq }\{\}$ from $a_2$ and $a_{21}$ and $sg_2$ and assms show $\text{thesis}$
      apply (simp add: ine-def, auto)  
      by (simp add: correctCompositionKS-exprChannel-s-Pex)
    qed
  next
  assume $a_{12}:\neg ine P (sE secret)$
  from this and $a_2$ and assms show $\text{thesis}$
    by (metis LocalSecretsComposition1 know.simps(2))
  qed

qed

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

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

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

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

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

next
fix secret
assume \(a2\) \(m = sKS\) secret
from \(\text{not know P and a2 have } sg1:\neg \text{ know } P (sKS\text{ secret}) \) by simp
then have \(sg1a: \neg \text{ ine } P (sE\text{ secret})\) by simp
from \(sg1\) have \(sg1b: sKS\text{ secret } \notin \text{ LocalSecrets } P\) by simp
from \(sg1\) and \(sg2\) and \(\text{sub } PQ\) and \(\text{cCompI have } sg3: \neg \text{ ine } PQ (sE\text{ secret})\)
by \((\text{rule know-composition-neg-ine-s})\)
from \(\text{sub } PQ\) and \(\text{cCompLoc and sg1a and sg2a and sg1b and sg2b have } sg4: sKS\text{ secret } \notin \text{ LocalSecrets } PQ\)
by \((\text{rule LocalSecretsComposition-neg1-s})\)
from \(sg3\) and \(sg4\) and \(a2\) show \(\text{thesis by simp}\)
\(\text{qed}\)

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

next
fix secret
assume a2: m = sKS secret
from this show \text{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) \lor ine Q (sE secret)
  by (simp add: TBtheorem1a)
  from this and a2 show \text{thesis} by auto
next
assume a22: \neg ine PQ (sE secret)
from this and knowPQ and a2 have sg5:
  sKS secret \in \text{LocalSecrets } PQ by auto
show \text{thesis}
proof (cases know Q m)
  assume know Q m
  from this show \text{thesis} by simp
next
assume not-knowQm: \neg know Q m
from not-knowQm and a2 have sg6a: \neg ine Q (sE secret) by simp
from not-knowQm and a2 have sg6b: sKS secret \notin \text{LocalSecrets } Q by simp
show \text{thesis}
proof (cases sKS secret \notin \text{LocalSecrets } P)
  assume sKS secret \notin \text{LocalSecrets } P
  from this and a2 show \text{thesis} by simp
next
assume sKS secret \notin \text{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 \text{thesis} by simp
qed
qed
qed

lemma eout-knows-nonKS-k:
assumes m \notin (\text{specKeys } A)
  and eout A (kE m)
and $\text{eoutKnowsECorrect } A (kE m)$
shows $\text{knows } A [kE m]$
using assms
by (metis Expression.distinct(1) Expression.inject(1) eoutKnowsECorrect-L1)

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

lemma not-knows-k-not-ine:
assumes $\neg \text{knows } A [kE m]$
shows $\neg \text{ine } A (kE m)$
using assms by (metis knows2know-neg-k not-know-k-not-ine)

lemma not-knows-s-not-ine:
assumes $\neg \text{knows } A [sE m]$
shows $\neg \text{ine } A (sE m)$
using assms by (metis knows2know-neg-s not-know-s-not-ine)

lemma not-knows-k-not-eout:
assumes $m \notin \text{specKeys } A$
and $\neg \text{knows } A [kE m]$
and $\text{eoutKnowsECorrect } A (kE m)$
shows $\neg \text{eout } A (kE m)$
using assms by (metis eout-knows-nonKS-k)

lemma not-knows-s-not-eout:
assumes $m \notin \text{specSecrets } A$
and $\neg \text{knows } A [sE m]$
and $\text{eoutKnowsECorrect } A (sE m)$
shows $\neg \text{eout } A (sE m)$
using assms by (metis eout-knows-nonKS-s)

lemma adv-not-knows1:
assumes $\text{out } P \subseteq \text{ins } A$
and $\neg \text{knows } A [kE m]$
shows $\neg \text{eout } P (kE m)$
using assms by (metis adv-not-know1 knows2know-neg-k)

lemma adv-not-knows2:
assumes $\text{out } P \subseteq \text{ins } A$
and $\neg \text{knows } A [sE m]$
shows $\neg \text{eout } P (sE m)$
using assms by (metis adv-not-know2 knows2know-neg-s)
lemma knows-decomposition-1-k:
assumes $kKS a \notin \text{specKeysSecrets } P$
and $kKS a \notin \text{specKeysSecrets } Q$
and subcomponents $PQ = \{P, Q\}$
and knows $PQ \[kE a\]$
and correctCompositionIn $PQ$
and correctCompositionLoc $PQ$
shows $\text{knows } P \[kE a\] \lor \text{knows } Q \[kE a\]$
using assms by (metis know-decomposition knows1k)

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

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

lemma knows-composition1-s:
assumes $(sKS m) \notin \text{specKeysSecrets } P$
and $(sKS m) \notin \text{specKeysSecrets } Q$
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) \notin \text{specKeysSecrets } P
and (kKS m) \notin \text{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) \notin \text{specKeysSecrets } P
and (sKS m) \notin \text{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 \notin \text{specKeysSecrets } P
and kKS m \notin \text{specKeysSecrets } Q
and \neg knows P [kE m]
and \neg knows Q [kE m]
and subcomponents PQ = \{P, Q\}
and correctCompositionLoc PQ
and correctCompositionIn PQ
and correctCompositionKS PQ
shows \neg knows PQ [kE m]
using assms by (metis know-decomposition knows1k)

lemma knows-composition-neg1-s:
assumes sKS m \notin \text{specKeysSecrets } P
and sKS m \notin \text{specKeysSecrets } Q
and \neg knows P [sE m]
and \neg knows Q [sE m]
and subcomponents PQ = \{P, Q\}
and correctCompositionLoc PQ
and correctCompositionIn PQ
and correctCompositionKS PQ
shows \neg 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 sg2a 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

40
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 \( \exists \ z1. \ a = kE z1 \))
  assume \( \exists \ z1. \ a = (kE z1) \)
  from this obtain z where a1: a = (kE z) by auto
  from a1 and Cons have sg4: (kKS z) \notin specKeysSecrets P
    by (simp add: notSpecKeysSecretsExpr-def)
  from a1 and Cons have sg5: (kKS z) \notin 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 \( \neg (\exists \ z1. \ a = kE z1) \)
  from this and Cons and sg3 have \( \exists \ z2. \ a = (sE z2) \) by auto
  from this obtain z where a2: a = (sE z) by auto
  from a2 and Cons have sg8: (sKS z) \notin specKeysSecrets P
    by (simp add: notSpecKeysSecretsExpr-def)
  from a2 and Cons have sg9: (sKS z) \notin 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 \lor knows Q e
  and subcomponents PQ = \{P, Q\}
  and correctCompositionIn PQ
  and correctCompositionKS PQ
  and \( \forall \ m. \ m \ mem e \longrightarrow ((\exists \ z. \ m = kE z) \lor (\exists \ z. \ m = sE z)) \)
and notSpecKeysSecretsExpr P e
and notSpecKeysSecretsExpr Q e
shows knows PQ e
using assms
by (metis knows-composition3 knows-composition4)
end

References

