Compositional properties of crypto-based components

Maria Spichkova

April 19, 2020

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)

derived

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**

```plaintext
cons
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**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
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
— 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
  EncrDecrKeys :: Keys ⇒ Keys ⇒ bool
  where
  ExtSign:
  EncrDecrKeys K1 K2 ⇒ (Ext K1 (Sign K2 E)) = E and
DecrEnc:
\[ \text{EncrDecrKeys } K1 \ K2 \rightarrow (\text{Decr } K2 \ (\text{Enc } K1 \ E)) = E \]

— Set of private keys of a component
\[ \text{consts} \]
\[ \text{specKeys} :: \text{specID} \Rightarrow \text{Keys set} \]
— Set of unguessable values used by a component
\[ \text{consts} \]
\[ \text{specSecrets} :: \text{specID} \Rightarrow \text{Secrets set} \]
— Join set of private keys and unguessable values used by a component
\[ \text{definition} \]
\[ \text{specKeysSecrets} :: \text{specID} \Rightarrow \text{KS set} \]
\[ \text{where} \]
\[ \text{specKeysSecrets } C \equiv \{ y . \exists x . y = (kKS x) \land (x \in (\text{specKeys } C))\} \cup \{ z . \exists s . z = (sKS s) \land (s \in (\text{specSecrets } C))\}\]

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

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

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

— 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
\[ \text{definition} \]
\[ \text{correctCompositionKS} :: \text{specID} \Rightarrow \text{bool} \]
where  
\( \text{correctCompositionKS} x \equiv \)  
\( \text{subcomponents} x \neq \{\} \rightarrow \)  
\( \text{specKeysSecrets} x = \bigcup (\text{specKeysSecrets} \setminus (\text{subcomponents} x)) \)

— Predicate defining set of correctness properties of the component’s  
— interface and relations on its private keys and unguessable values

definition  
\( \text{correctComponentSecrecy} :: \text{specID} \Rightarrow \text{bool} \)

where  
\( \text{correctComponentSecrecy} x \equiv \)  
\( \text{correctCompositionKS} x \land \)  
\( \text{correctCompositionSecrets} x \land \)  
\( \text{correctCompositionKeys} x \land \)  
\( \text{correctCompositionLoc} x \land \)  
\( \text{correctCompositionIn} x \land \)  
\( \text{correctCompositionOut} x \land \)  
\( \text{correctInOutLoc} x \)

— Predicate \( \text{exprChannel} I \ E \) defines whether the expression item \( E \) can be sent  
via the channel \( I \)

definition  
\( \text{exprChannel} :: \text{chanID} \Rightarrow \text{Expression} \Rightarrow \text{bool} \)

— Predicate \( \text{eoutM} sP M E \) 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  
\( \text{eoutM} :: \text{specID} \Rightarrow \text{chanID set} \Rightarrow \text{Expression} \Rightarrow \text{bool} \)

where  
\( \text{eoutM} sP M E \equiv \)  
\( \exists (ch :: \text{chanID}). ((ch \in (\text{out} sP)) \land (\text{exprChannel} ch E)) \)

— Predicate \( \text{eout} sP E \) 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  
\( \text{eoutM} :: \text{specID} \Rightarrow \text{chanID set} \Rightarrow \text{Expression} \Rightarrow \text{bool} \)

where  
\( \text{eoutM} sP M E \equiv \)  
\( \exists (ch :: \text{chanID}). ((ch \in (\text{out} sP)) \land (ch \in M) \land (\text{exprChannel} ch E)) \)

— Predicate \( \text{ineM} sP M E \) 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  
\( \text{ine} :: \text{specID} \Rightarrow \text{Expression} \Rightarrow \text{bool} \)
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 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
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**

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

**where**

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

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

\[
\land
\]

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

— 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 l`

**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 \notin loc PQ`

— and `ch \in ins P`

**shows** `ch \in ins PQ`

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

— If a channel belongs to the set of input channels of the composition 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 ins PQ`

**shows** `(ch \in ins P) \lor (ch \in 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 correctCompositionKeys \( C \)
and \( x \in \text{subcomponents} \ C \)
and \( xb \in \text{specKeys} \ C \)
shows \( \exists x \in \text{subcomponents} C \cdot (xb \in \text{specKeys} x) \)
using assms by (simp add: correctCompositionKeys-def, auto)

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

lemma correctCompositionKeys-subcomp2:
assumes 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 correctCompositionSecrets \( C \)
and \( xb \in \text{subcomponents} \ C \)

9
and \( x_c \in \text{specSecrets} \) zb
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 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
                correctCompositionKeys-def
                correctCompositionSecrets-def
                specKeysSecrets-def, blast)
lemma correctCompositionKS-subcomp1:
assumes correctCompositionKS C and h1: x ∈ subcomponents C and xa ∈ specKeys C
shows ∃ y ∈ subcomponents C. (xa ∈ 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 ∃ y ∈ subcomponents C. xa ∈ 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 \( ks \in \text{specKeysSecrets} \ PQ \)
sounds \( ks \in \text{specKeysSecrets} \ P \lor ks \in \text{specKeysSecrets} \ Q \)
using assms by (simp add: correctCompositionKS-def)

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

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

lemma correctCompositionKS-negQ:
assumes subcomponents PQ = \{P, Q\}
    and correctCompositionKS PQ
    and ks /\in specKeysSecrets PQ
shows ks /\in 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: ineM-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: eoutM-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 ¬ (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
\textbf{and} subcomponents $PQ = \{P,Q\}$
\textbf{and} correctCompositionIn $PQ$
\textbf{and} $\exists$ ch. $(ch \in (\text{ins } P) \land exprChannel ch E \land ch \notin (\text{loc } PQ))$
\textbf{shows} ine $PQ E$
\textbf{using} \texttt{assms} by (simp add: ine-def correctCompositionIn-def, auto)

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

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

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

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

\textit{theorem} TBtheorem4b-notP1:
\textbf{assumes} ineM $P M E$
\textbf{and} $\neg$ ineM $Q M E$
\textbf{and} subcomponents $PQ = \{P,Q\}$
\textbf{and} correctCompositionIn $PQ$
and \( \exists \ ch. (\text{ine-exprChannelSingle} \ P \ ch \ E) \land (ch \in M) \land (ch \in (\text{loc} \ PQ)) \) shows \( \neg \text{ine} \ M \ PQ \ M \ E \) 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 ine-exprChannelSet \( P \ ChSet \ E \)
and \( \forall (x ::\text{chanID}). ((x \in ChSet) \rightarrow (x \in (\text{loc} \ PQ))) \)
shows \( \neg \text{ine} \ PQ \ E \) using assms by (simp add: ine-def correctCompositionIn-def ine-exprChannelSet-def, auto)

**Theorem TBtheorem4b-notPQ:**
assumes \( \neg \text{ine} \ M \ PQ \ M \ E \)
and subcomponents \( PQ = \{P, Q\} \)
and correctCompositionIn \( PQ \)
and ine-exprChannelSet \( P \ ChSet \ E \)
and \( \forall (x ::\text{chanID}). ((x \in ChSet) \rightarrow (x \in (\text{loc} \ PQ))) \)
and \( \forall (x ::\text{chanID}). ((x \in ChSetP) \rightarrow (x \in (\text{loc} \ PQ))) \)
and \( \forall (x ::\text{chanID}). ((x \in ChSetQ) \rightarrow (x \in (\text{loc} \ PQ))) \)
shows \( \neg \text{ine} \ PQ \ E \) using assms by (simp add: ine-def correctCompositionIn-def ine-exprChannelSet-def, auto)

**Lemma ineM-Un1:**
assumes \( \text{ine} \ M \ P \ A \ E \)
shows \( \text{ine} \ M \ (P \ Un \ B) \ E \)
using assms by (simp add: ineM-def, auto)

**Theorem TBtheorem4b-notPQ:**
assumes subcomponents \( PQ = \{P, Q\} \)
and correctCompositionIn \( PQ \)
and ine-exprChannelSet P ChSet P E
and ine-exprChannelSet Q ChSet Q E
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 \( \neg \text{ineM PQ M E} \)
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)

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

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

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

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

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

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

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

**Theorem TBtheorem5a-notP2:**
assumes $\neg \text{eout } Q \ E$
and subcomponents $PQ = \{P, Q\}$
and $\text{correctCompositionOut } PQ$
and $\text{out-exprChannelSet } P \ ChSet \ E$
and $\forall \ (x :: \text{chanID}). ((x \in \text{ChSet}) \longrightarrow (x \in \text{loc } PQ))$
signifies $\neg \text{eout } PQ \ E$
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 \(\forall (x :: \text{chanID}). ((x \in \text{ChSet}) \rightarrow (x \in (\text{loc PQ})))\)

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

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

**Shows** ¬ 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 \(M = \text{ChSetP} \cup \text{ChSetQ}\)

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** ¬ 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 $\Rightarrow$ KS set

where

LocalSecretsDef:
LocalSecrets A =
\{(m :: KS), m \notin specKeysSecrets A \land
(\exists x y. ((x \in loc A) \land m = (kKS y) \land (exprChannel x (kE y))))
| (\exists x z. ((x \in loc A) \land m = (sKS z) \land (exprChannel x (sE z)) ) ) \} \cup (\bigcup (LocalSecrets' (subcomponents A )))

lemma LocalSecretsComposition1:
assumes ls \in LocalSecrets P
and subcomponents PQ = \{P, Q\}
sows ls \in LocalSecrets PQ
using assms by (simp (no-asn) only: LocalSecretsDef, auto)

lemma LocalSecretsComposition-exprChannel-k:
assumes exprChannel x (kE Keys)
and \neg ine P (kE Keys)
and \neg ine Q (kE Keys)
and \neg (x \notin ins P \land x \notin ins Q)
sows False
using assms by (metis ine-def)

lemma LocalSecretsComposition-exprChannel-s:
assumes exprChannel x (sE Secrets)
and \neg ine P (sE Secrets)
and \neg ine Q (sE Secrets)
and \neg (x \notin ins P \land x \notin ins Q)
sows False
using assms by (metis ine-ins-neg1)

lemma LocalSecretsComposition-neg1-k:
assumes subcomponents PQ = \{P, Q\}
and correctCompositionLoc PQ
and \neg ine P (kE Keys)
and \neg ine Q (kE Keys)
and kKS Keys \notin LocalSecrets P
and kKS Keys \notin LocalSecrets Q
sows kKS Keys \notin 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 \((kKS\ m)\notin\text{specKeysSecrets}\ P\)
and \((kKS\ m)\notin\text{specKeysSecrets}\ Q\)
and \(\neg\text{ine}\ P\ (kE\ m)\)
and \(\neg\text{ine}\ Q\ (kE\ m)\)
and \((kKS\ m)\notin((\text{LocalSecrets}\ P)\cup(\text{LocalSecrets}\ Q))\)
shows \((kKS\ m)\notin(\text{LocalSecrets}\ PQ)\)

proof –
from \text{assms}\ show \(\neg\text{thesis}\)
apply (simp (no-asms) only: LocalSecretsDef,
 simp add: correctCompositionLoc-def, clarify)
by (rule 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 \(\text{notKSP}: (sKS\ m)\notin\text{specKeysSecrets}\ P\)
and \(\text{notKSQ}: (sKS\ m)\notin\text{specKeysSecrets}\ Q\)
and \(\neg\text{ine}\ P\ (sE\ m)\)
and \(\neg\text{ine}\ Q\ (sE\ m)\)
and \(\text{notLocSeqPQ}: (sKS\ m)\notin((\text{LocalSecrets}\ P)\cup(\text{LocalSecrets}\ Q))\)
shows \((sKS\ m)\notin(\text{LocalSecrets}\ PQ)\)

proof –
from \text{subPQ}\ and \text{cCompKS}\ and \text{notKSP}\ and \text{notKSQ}\ have \(sg1: sKS\ m\notin\text{specKeysSecrets}\ PQ\)
by (simp add: correctCompositionKS-neg1)
from \text{subPQ}\ and \text{cCompLoc}\ and \text{notLocSeqPQ}\ have \(sg2: sKS\ m\notin(\text{LocalSecrets}\ \cup\ \text{subcomponents}\ PQ)\)
by simp
from \(sg1\) and \(sg2\) and \text{assms}\ show \(\neg\text{thesis}\)
apply (simp (no-asms) only: LocalSecretsDef,
 simp add: correctCompositionLoc-def, clarify)
by (rule 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 \(h1:\forall\ m .\ ks = kKS\ m\longrightarrow(\neg\text{ine}\ P\ (kE\ m)\land\neg\text{ine}\ Q\ (kE\ m))\)
and \(h2:\forall\ m .\ ks = sKS\ m\longrightarrow(\neg\text{ine}\ P\ (sE\ m)\land\neg\text{ine}\ Q\ (sE\ m))\)
and \(ks\notin((\text{LocalSecrets}\ P)\cup(\text{LocalSecrets}\ Q))\)
shows \(ks\notin(\text{LocalSecrets}\ PQ)\)

proof (cases \(ks\))
fix \(m\)
assume $a_1: ks = kKS m$
from this and $h_1$ have $\neg \text{ine } P (kE m) \land \neg \text{ine } Q (kE m)$ by simp
from this and $a_1$ and assms show $\text{thesis}$
  by (simp add: LocalSecretsComposition-neg-k)

next
fix $m$
assume $a_2: ks = sKS m$
from this and $h_2$ have $\neg \text{ine } P (sE m) \land \neg \text{ine } Q (sE m)$ by simp
from this and $a_2$ and assms show $\text{thesis}$
  by (simp add: LocalSecretsComposition-neg-s)
qed

lemma LocalSecretsComposition-neg1-s:
assumes subcomponents $PQ = \{P, Q\}$
and correctCompositionLoc $PQ$
and $\neg \text{ine } P (sE s)$
and $\neg \text{ine } Q (sE s)$
and $sKS s \notin \text{LocalSecrets } P$
and $sKS s \notin \text{LocalSecrets } Q$
shows $sKS s \notin \text{LocalSecrets } PQ$
proof (cases ks)
fix $m$
assume $a_1: ks = kKS m$
from this and $h_1$ have $\neg \text{ine } P (kE m) \land \neg \text{ine } Q (kE m)$ by simp
from this and $a_1$ and assms show $\text{thesis}$
  by (simp add: LocalSecretsComposition-neg1-k)
next
fix $m$
assume $a_2: ks = sKS m$
from this and $h_2$ have $\neg \text{ine } P (sE m) \land \neg \text{ine } Q (sE m)$ by simp
from this and $a_2$ and assms show $\text{thesis}$
qed

lemma LocalSecretsComposition-neg1-
assumes subcomponents $PQ = \{P, Q\}$
and correctCompositionLoc $PQ$
and $h_1: \forall m. ks = kKS m \rightarrow (\neg \text{ine } P (kE m) \land \neg \text{ine } Q (kE m))$
and $h_2: \forall m. ks = sKS m \rightarrow (\neg \text{ine } P (sE m) \land \neg \text{ine } Q (sE m))$
and $ks \notin \text{LocalSecrets } P$
and $ks \notin \text{LocalSecrets } Q$
shows $ks \notin \text{LocalSecrets } PQ$
proof (cases ks)
fix $m$
assume $a_1: ks = kKS m$
from this and $h_1$ have $\neg \text{ine } P (kE m) \land \neg \text{ine } Q (kE m)$ by simp
from this and $a_1$ and assms show $\text{thesis}$
  by (simp add: LocalSecretsComposition-neg1-k)
next
fix $m$
assume $a_2: ks = sKS m$
from this and $h_2$ have $\neg \text{ine } P (sE m) \land \neg \text{ine } Q (sE m)$ by simp
from this and $a_2$ and assms show $\text{thesis}$

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 /∈ LocalSecrets P
  and exprChannel ch (sE secret)
  and sKS secret /∈ specKeysSecrets P
shows ch /∈ 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 /∈ LocalSecrets PQ
  and ch ∈ ins P
  and exprChannel ch (kE key)
  and kKS key /∈ specKeysSecrets PQ
  and correctCompositionIn PQ
shows ch ∈ ins PQ ∧ 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 /∈ LocalSecrets PQ
  and ch ∈ ins P
  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-P)

lemma correctCompositionKS-exprChannel-k-Q:
assumes subcomponents PQ = {P, Q}
  and correctCompositionKS PQ
  and kKS key /∈ LocalSecrets PQ
  and ch ∈ ins Q
  and h1: exprChannel ch (kE key)
  and kKS key /∈ specKeysSecrets PQ
  and correctCompositionIn PQ
shows ch ∈ ins PQ ∧ exprChannel ch (kE key)
proof –
  from assms have ch /∈ loc PQ
    by (simp add: LocalSecretsComposition-neg-loc-k)
  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-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 \text{hl:} 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 \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**

\begin{align*}
\text{knows} & : \text{specID} \Rightarrow \text{Expression list} \Rightarrow \text{bool} \\
\text{where} \\
\text{knows-empty-expression} & : \\
\text{knows} C \[] = \text{True} \text{ and} \\
k\text{now1} & : \\
\text{knows} C [\text{KS2Expression} (kKS m1)] = \text{know} C (kKS m1) \text{ and} \\
k\text{now2} & : \\
\text{knows} C [\text{KS2Expression} (sKS m2)] = \text{know} C (sKS m2) \text{ and} \\
k\text{now3} & : \\
\text{knows} A (e1 \oplus e) \rightarrow \text{knows} A e \text{ and} \\
k\text{now4} & : \\
\text{knows} A (e \oplus e1) \rightarrow \text{knows} A e \text{ and} \\
k\text{now5} & : \\
(\text{knows} A c1) \land (\text{knows} A e2) \rightarrow \text{knows} A (e1 \oplus e2) \text{ and} \\
k\text{now6} & : \\
(\text{IncrDecrKeys} k1 k2) \land (\text{know} A (kKS k1)) \land (\text{knows} A (\text{Enc} k1 e)) \rightarrow \text{knows} A e \\
\text{and} \\
k\text{now7} & : \\
(\text{know} A (kKS k)) \land (\text{knows} A e1) \rightarrow \text{knows} A (\text{Enc} k e1) \\
\text{and} \\
k\text{now8} & : \\
(\text{knows} A (sKS m)) \land (\text{knows} A e) \rightarrow \text{knows} A (\text{Sign} k e1) \\
\text{and} \\
k\text{now9} & : \\
(\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))
\end{align*}

**lemma** **eoutKnowCorrect-L1k**: 

\begin{align*}
\text{primrec} \\
eoutKnowCorrect : \text{specID} \Rightarrow \text{KS} \Rightarrow \text{bool} \\
\text{where} \\
eout-know-k & : \\
eoutKnowCorrect C (kKS m) = \\
(\langle eout C (kE m) \rangle \leftarrow (m \in (\text{specKeys } C) \lor (\text{know} C (kKS m)))) \mid \\
eout-know-s & : \\
eoutKnowCorrect C (sKS m) = \\
(\langle eout C (sE m) \rangle \leftarrow (m \in (\text{specSecrets } C) \lor (\text{know} C (sKS m))))
\end{align*}

**definition** **eoutKnowsECorrect** : \text{specID} \Rightarrow \text{Expression} \Rightarrow \text{bool} 

**where** 

\begin{align*}
eoutKnowsECorrect C e & \equiv \\
(\langle eout C e \rangle \leftarrow \\
(\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)))
\end{align*}

**lemma** **eoutKnowCorrect-L1k**: 

27
assumes eout\text{KnowCorrect} C (kKS m) 
and eout C (kE m) 
shows m \in (\text{specKeys C}) \lor (\text{know C} \ (kKS m)) 
using assms by (metis eout-know-k)

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

lemma eout\text{KnowsECorrect-L1s}: 
assumes eout\text{KnowsECorrect} C e 
and 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 know\text{2knows-k}: 
assumes know A (kKS m) 
shows knows A [kE m] 
using assms 
by (metis KS2Expression.simps(1) know1k)

lemma know\text{2know-k}: 
assumes knows A [kE m] 
shows know A (kKS m) 
using assms 
by (metis KS2Expression.simps(1) know1k)

lemma know\text{2knowsPQ-k}: 
assumes know P \ (kKS m) \lor know Q \ (kKS m) 
shows knows P [kE m] \lor knows Q [kE m] 
using assms by (metis know2knows-k)

lemma know\text{2knowPQ-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 know2know-k)

lemma know\text{1k}: 
know A \ (kKS m) = knows A \ [kE m] 
by (metis know2knows-k knows2know-k)

lemma know\text{2knows-neg-k}: 
assumes \neg know A \ (kKS m) 
shows \neg 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 knows2knowsPQ-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 knows1s:
know A \((sKS m)\) = knows A \([sE m]\)
by (metis knows2knows-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 \(e2 = e1 @ e \lor e2 = e @ e1\)
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 \( \forall \ x, \ x \in \text{ins} \ PQ \rightarrow \neg \text{exprChannel} \ x \ m \)
shows   ch : loc PQ
using    assms by (simp add: correctCompositionIn-def, auto)


lemma eout-know-nonKS-k:
assumes m /\ 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 /\ 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 /\ 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 /\ 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 \( \neg \text{out } P \ (kE \ m) \)
using assms
by (metis (full-types) out-def ine-ins-neg1 not-know-k-not-ine rev-subsetD)

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

lemma LocalSecrets-L1:
assumes \((kKS) \text{ key } \in \text{LocalSecrets } P\)
and \((kKS \text{ key }) \notin \bigcup (\text{LocalSecrets } \cup \text{ subcomponents } P)\)
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 \(kKS \text{ key } \in \bigcup (\text{LocalSecrets } \cup \text{ subcomponents } P)\)
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: \(\text{subcomponents } PQ = \{P, Q\}\)
and cCompI: \(\text{correctCompositionIn } PQ\)
and cCompKS: \(\text{correctCompositionKS } PQ\)
shows \(\text{know } PQ \ m\)
proof (cases \(m\))
  fix \(key\)
  assume a1: \(m = kKS \text{ key}\)
  show ?thesis
  proof (cases \(\text{ine } P \ (kE \text{ key})\))
    assume a11: \(\text{ine } P \ (kE \text{ key})\)
    from this have a11text: \(\text{ine } P \ (kE \text{ key}) \mid \text{ine } Q \ (kE \text{ key})\) by simp
    from subPQ and cCompKS and notKSP and notKSQ
    have \(m \notin \text{specKeysSecrets } PQ\)
    by (rule correctCompositionKS-neg1)
    from this and a1 have sg1: \(kKS \text{ key } \notin \text{specKeysSecrets } PQ\) by simp
    from a1 and a11text and cCompKS show ?thesis
  proof (cases loc \(PQ = \{\}\))
    assume a1locE: \(\text{loc } PQ = \{\}\)
    from a11text and subPQ and cCompI and a1locE have \(\text{ine } PQ \ (kE \text{ key})\)
    by (rule TBtheorem4a-empty)
    from this and a1 show ?thesis by auto
  next

31
assume \( a_{11} \text{locNE}: \text{loc } PQ \neq \{ \} \)
from \( a_1 \) and \( a_{11} \) and \( s_{1} \) and asms show \(?thesis \\
apply (simp add: ine-def, auto) 
by (simp add: correctCompositionKS-exprChannel-k-Pex)
qed

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

next
fix secret
assume \( a_{2}: m = s_{KS} \text{ secret} \)
show \(?thesis \\
proof (cases \text{ ine } P (s_{E} \text{ secret})) 
assume \( a_{21}: \text{ine } P (s_{E} \text{ secret}) \)
from this have \( a_{21\text{text}: \text{ine } P (s_{E} \text{ secret}) \mid \text{ine } Q (s_{E} \text{ secret})} \) by simp
from subPQ and cCompKS and notKSP and notKSQ have \( m \notin \text{specK-}
\text{eysSecrets } PQ \)
by (rule correctCompositionKS-neg1)
from this and \( a_2 \) have \( s_{2}: s_{KS} \text{ secret} \notin \text{specKeysSecrets } PQ \) by simp
from \( a_2 \) and \( a_{2\text{text}} \) and cCompKS show \(?thesis \\
proof (cases \text{loc } PQ = \{ \}) 
assume \( a_{21\text{locE}: \text{loc } PQ = \{ } \)
from \( a_{21\text{text}} \) and subPQ and cCompI and \( a_{21\text{locE}} \) have \( \text{ine } PQ (s_{E} \text{ secret}) \)
by (rule TBtheorem4a-empty)
from this and \( a_2 \) show \(?thesis \) by auto
next
assume \( a_{21\text{locNE}: \text{loc } PQ \neq \{ } \)
from \( a_2 \) and \( a_{21} \) and \( s_{2} \) and asms show \(?thesis \\
apply (simp add: ine-def, auto) 
by (simp add: correctCompositionKS-exprChannel-s-Pex)
qed

next
assume \( a_{12}: \neg \text{ine } P (s_{E} \text{ secret}) \)
from this and \( a_2 \) and asms show \(?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
\[ 32 \]
using assms by (metis insert-commute know-composition1)

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

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

lemma know-composition-neg1:
assumes \( \neg \text{know } P \ m \)
and \( \neg \text{know } Q \ m \)
and \( \text{subPQ} : \text{subcomponents } PQ = \{P, Q\} \)
and \( \text{cCompLoc} : \text{correctCompositionLoc } PQ \)
and \( \text{cCompI} : \text{correctCompositionIn } PQ \)
shows \( \neg \text{know } PQ \ m \)
proof (cases \( m \))
  fix key
  assume a1:\( m = kKS \text{ key} \)
  from \( \neg \text{know } P \ m \) and a1 have sg1:\( \neg \text{know } P \ (kKS \text{ key}) \) by simp
  then have sg1a:\( \neg \text{ine } P \ (kE \text{ key}) \) by simp
  from sg1 have sg1b:\( kKS \text{ key } \notin \text{LocalSecrets } P \) by simp
  from \( \neg \text{known } Q \) and a1 have sg2:\( \neg \text{know } Q \ (kKS \text{ key}) \) by simp
  then have sg2a:\( \neg \text{ine } Q \ (kE \text{ key}) \) by simp
  from sg2 have sg2b:\( kKS \text{ key } \notin \text{LocalSecrets } Q \) by simp
  from sg1 and sg2 and subPQ and cCompI have sg3:\( \neg \text{ine } PQ \ (kE \text{ key}) \)
  by (rule know-composition-neg-ine-k)
  from subPQ and cCompLoc and sg1a and sg2a and sg1b and sg2b have sg4:\( kKS \text{ key } \notin \text{LocalSecrets } PQ \)
  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 \notin 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 \notin 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 \notin 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 \lor 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) \lor 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 \in 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 \notin LocalSecrets Q by simp
show ?thesis
proof (cases kKS key \in LocalSecrets P)
assume kKS key \in LocalSecrets P
from this and a1 show ?thesis by simp
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 ?thesis by simp
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) $\lor$ ine $Q$ ($sE$ secret)
by (simp add: TBtheorem1a)
from this and $a2$ show ?thesis by auto
next
assume $a22$: $\neg$ ine $PQ$ ($sE$ secret)
from this and $knowPQ$ and $a2$ have $sg5$: $sKS$ secret $\in$ LocalSecrets $PQ$ by auto
show ?thesis
proof (cases $know Q$ $m$)
assume $know Q$ $m$
from this show ?thesis by simp
next
assume $not-knowQm$: $\neg$ $know Q$ $m$
from $not-knowQm$ and $a2$ have $sg6a$: $\neg$ ine $Q$ ($sE$ secret) by simp
from $not-knowQm$ and $a2$ have $sg6b$: $sKS$ secret $\notin$ LocalSecrets $Q$ by simp
show ?thesis
proof (cases $sKS$ secret $\notin$ LocalSecrets $P$)
assume $sKS$ secret $\notin$ LocalSecrets $P$
from this and $a2$ show ?thesis by simp
next
assume $sKS$ secret $\notin$ 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
next
lemma $eout-knows-nonKS-k$:
assumes $m$ $\notin$ (specKeys $A$)
and $eout A$ ($kE$ $m$)
and \( \text{eoutKnowsECorrect} \ A \ (kE \ m) \)
shows knows \( A \ [kE \ m] \)
using assms by (metis \( \text{Expression.distinct}(1) \) \( \text{Expression.inject}(1) \) \( \text{eoutKnowsECorrect-L1} \))

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

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

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

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

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

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

lemma \( \text{adv-not-knows2} \):
assumes \( \text{out} \ P \subseteq \text{ins} \ A \)
and \( \neg \) knows \( A \ [sE \ m] \)
shows \( \neg \) eout \( P \ (sE \ m) \)
using assms by (metis \( \text{adv-not-know2} \) \( \text{knows2know-neg-s} \))
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 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 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 \( \forall z. a = kE z \rightarrow kKS z \notin \text{specKeysSecrets } P \land kKS z \notin \text{specKeysSecrets } Q \)
and \( \forall z. a = sE z \rightarrow sKS z \notin \text{specKeysSecrets } P \land sKS z \notin \text{specKeysSecrets } Q \)
shows knows \( P [a] \lor \text{knows } Q [a] \)
using assms
by (metis know-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 correctCompositionKS \( PQ \)
shows 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) \not\in specKeysSecrets P
and (kKS m) \not\in 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) \not\in specKeysSecrets P
and (sKS m) \not\in 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 \not\in specKeysSecrets P
and kKS m \not\in 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 \not\in specKeysSecrets P
and sKS m \not\in 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]
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 \( \neg \) knows P (a # e)  
and knows P [a]  
shows \( \neg \) knows P e  
using assms by (metis knows-concat-3)

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

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

lemma not-knows-tail-not-knows-conc:  
assumes \( \neg \) knows P e  
shows \( \neg \) 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 : correctCompositionI in PQ  
and cCompKS : correctCompositionKS PQ  
and \( \forall (m :: Expression). ((m mem e) \rightarrow ((\exists z1. m = (kE z1)) \lor (\exists z2. m = (sE z2))))\)  
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 \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 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 \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 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

lemma knows-composition4:
  assumes knows Q e
    and subPQ: subcomponents PQ = \{P, Q\}
    and cCompI: correctCompositionIn PQ
    and cCompKS: correctCompositionKS PQ
    and \forall m. m mem e \rightarrow ((\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
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

