Compositional properties of crypto-based components

Maria Spichkova

May 21, 2019

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

theory *Secrecy-types*
imports *Main*
begin

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

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

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

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

end

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

theory *inout*
imports *Secrecy-types*
begin

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

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

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

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

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

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

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

— Predicate insuring the correct relations between

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

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

— If a component is an elementary one (has no subcomponents)
— its set of local channels should be empty
lemma *subcomponents-loc*:
assumes *correctCompositionLoc x*
and *subcomponents x* = {}
shows *loc x* = {}
⟨*proof*⟩

end

3 Secrecy: Definitions and properties

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

— Encryption, decryption, signature creation and signature verification functions
— For these functions we define only their signatures and general axioms,
— because in order to reason effectively, we view them as abstract functions and
— abstract from their implementation details
consts
Enc :: *Keys* ⇒ *Expression list* ⇒ *Expression list*
Decr :: *Keys* ⇒ *Expression list* ⇒ *Expression list*
Sign :: *Keys* ⇒ *Expression list* ⇒ *Expression list*
Ext :: *Keys* ⇒ *Expression list* ⇒ *Expression list*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

— If a list of expression items does not contain any private key
— or unguessable value of a component P, then the first element
— of the list is neither a private key nor unguessable value of P
lemma *notSpecKeysSecretsExpr-L1*:
assumes *notSpecKeysSecretsExpr P* (*a* # *l*)
shows *notSpecKeysSecretsExpr P* [*a*]
⟨*proof*⟩
lemma *notSpecKeysSecretsExpr-L2*:
assumes *notSpecKeysSecretsExpr P* (*a* # *l*)
shows *notSpecKeysSecretsExpr P l*
⟨*proof*⟩
lemma *correctCompositionIn-L1*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ch* ∉ *loc PQ*
and *ch* ∈ *ins P*
shows *ch* ∈ *ins PQ*
⟨*proof*⟩
lemma *correctCompositionIn-L2*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ch* ∈ *ins PQ*
shows (*ch* ∈ *ins P*) ∨ (*ch* ∈ *ins Q*)
⟨*proof*⟩

lemma *ineM-L1*:
assumes *ch* ∈ *M*
and *ch* ∈ *ins P*
and *exprChannel ch E*
shows *ineM P M E*
⟨*proof*⟩

lemma *ineM-ine*:
assumes *ineM P M E*
shows *ine P E*
⟨*proof*⟩

lemma *not-ine-ineM*:

assumes ¬ *ine P E*
shows ¬ *ineM P M E*
⟨*proof*⟩

lemma *eoutM-eout*:
assumes *eoutM P M E*
shows *eout P E*
⟨*proof*⟩

lemma *not-eout-eoutM*:
assumes ¬ *eout P E*
shows ¬ *eoutM P M E*
⟨*proof*⟩

lemma *correctCompositionKeys-subcomp1*:
assumes *correctCompositionKeys C*
and *x* ∈ *subcomponents C*
and *xb* ∈ *specKeys C*
shows ∃ *x* ∈ *subcomponents C*. (*xb* ∈ *specKeys x*)
⟨*proof*⟩

lemma *correctCompositionSecrets-subcomp1*:
assumes *correctCompositionSecrets C*
and *x* ∈ *subcomponents C*
and *s* ∈ *specSecrets C*
shows ∃ *x* ∈ *subcomponents C*. (*s* ∈ *specSecrets x*)
⟨*proof*⟩

lemma *correctCompositionKeys-subcomp2*:
assumes *correctCompositionKeys C*
and *xb* ∈ *subcomponents C*
and *xc* ∈ *specKeys xb*
shows *xc* ∈ *specKeys C*
⟨*proof*⟩

lemma *correctCompositionSecrets-subcomp2*:
assumes *correctCompositionSecrets C*
and *xb* ∈ *subcomponents C*
and *xc* ∈ *specSecrets xb*
shows *xc* ∈ *specSecrets C*
⟨*proof*⟩

lemma *correctCompKS-Keys*:
assumes *correctCompositionKS C*
shows *correctCompositionKeys C*
⟨*proof*⟩

lemma *correctCompKS-Secrets*:
assumes *correctCompositionKS C*

shows *correctCompositionSecrets C*
⟨*proof*⟩

lemma *correctCompKS-KeysSecrets*:
assumes *correctCompositionKeys C*
and *correctCompositionSecrets C*
shows *correctCompositionKS C*
⟨*proof*⟩

lemma *correctCompositionKS-subcomp1*:
assumes *correctCompositionKS C*
and *h1*:$x \in$ *subcomponents C*
and $xa \in$ *specKeys C*
shows $\exists\ y \in$ *subcomponents C*. ($xa \in$ *specKeys y*)
⟨*proof*⟩

lemma *correctCompositionKS-subcomp2*:
assumes *correctCompositionKS C*
and *h1*:$x \in$ *subcomponents C*
and $xa \in$ *specSecrets C*
shows $\exists\ y \in$ *subcomponents C*. $xa \in$ *specSecrets y*
⟨*proof*⟩

lemma *correctCompositionKS-subcomp3*:
assumes *correctCompositionKS C*
and $x \in$ *subcomponents C*
and $xa \in$ *specKeys x*
shows $xa \in$ *specKeys C*
⟨*proof*⟩

lemma *correctCompositionKS-subcomp4*:
assumes *correctCompositionKS C*
and $x \in$ *subcomponents C*
and $xa \in$ *specSecrets x*
shows $xa \in$ *specSecrets C*
⟨*proof*⟩

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

lemma *correctCompositionKS-neg1*:
assumes *subcomponents PQ* = $\{P, Q\}$
and *correctCompositionKS PQ*
and $ks \notin$ *specKeysSecrets P*
and $ks \notin$ *specKeysSecrets Q*

shows $ks \notin specKeysSecrets\ PQ$
⟨*proof*⟩

lemma *correctCompositionKS-negP*:
assumes $subcomponents\ PQ = \{P,\ Q\}$
and $correctCompositionKS\ PQ$
and $ks \notin specKeysSecrets\ PQ$
shows $ks \notin specKeysSecrets\ P$
⟨*proof*⟩

lemma *correctCompositionKS-negQ*:
assumes $subcomponents\ PQ = \{P,\ Q\}$
and $correctCompositionKS\ PQ$
and $ks \notin specKeysSecrets\ PQ$
shows $ks \notin specKeysSecrets\ Q$
⟨*proof*⟩

lemma *out-exprChannelSingle-Set*:
assumes $out\text{-}exprChannelSingle\ P\ ch\ E$
shows $out\text{-}exprChannelSet\ P\ \{ch\}\ E$
⟨*proof*⟩

lemma *out-exprChannelSet-Single*:
assumes $out\text{-}exprChannelSet\ P\ \{ch\}\ E$
shows $out\text{-}exprChannelSingle\ P\ ch\ E$
⟨*proof*⟩

lemma *ine-exprChannelSingle-Set*:
assumes $ine\text{-}exprChannelSingle\ P\ ch\ E$
shows $ine\text{-}exprChannelSet\ P\ \{ch\}\ E$
⟨*proof*⟩

lemma *ine-exprChannelSet-Single*:
assumes $ine\text{-}exprChannelSet\ P\ \{ch\}\ E$
shows $ine\text{-}exprChannelSingle\ P\ ch\ E$
⟨*proof*⟩

lemma *ine-ins-neg1*:
assumes $\neg\ ine\ P\ m$
and $exprChannel\ x\ m$
shows $x \notin ins\ P$
⟨*proof*⟩

theorem *TBtheorem1a*:
assumes $ine\ PQ\ E$
and $subcomponents\ PQ = \{P,Q\}$
and $correctCompositionIn\ PQ$
shows $ine\ P\ E\ \vee\ ine\ Q\ E$
⟨*proof*⟩

theorem *TBtheorem1b*:
assumes *ineM PQ M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
shows *ineM P M E* ∨ *ineM Q M E*
⟨*proof*⟩

theorem *TBtheorem2a*:
assumes *eout PQ E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
shows *eout P E* ∨ *eout Q E*
⟨*proof*⟩

theorem *TBtheorem2b*:
assumes *eoutM PQ M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
shows *eoutM P M E* ∨ *eoutM Q M E*
⟨*proof*⟩

lemma *correctCompositionIn-prop1*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *x* ∈ (*ins PQ*)
shows (*x* ∈ (*ins P*)) ∨ (*x* ∈ (*ins Q*))
⟨*proof*⟩

lemma *correctCompositionOut-prop1*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *x* ∈ (*out PQ*)
shows (*x* ∈ (*out P*)) ∨ (*x* ∈ (*out Q*))
⟨*proof*⟩

theorem *TBtheorem3a*:
assumes ¬ (*ine P E*)
and ¬ (*ine Q E*)
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
shows ¬ (*ine PQ E*)
⟨*proof*⟩

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*:$ch \in M$
and *chPQ*:$ch \in ins\ PQ$
and *eCh*:*exprChannel ch E*
shows *False*
⟨*proof*⟩

theorem *TBtheorem3b*:
assumes $\neg\ (ineM\ P\ M\ E)$
and $\neg\ (ineM\ Q\ M\ E)$
and $subcomponents\ PQ = \{P,Q\}$
and *correctCompositionIn PQ*
shows $\neg\ (ineM\ PQ\ M\ E)$
⟨*proof*⟩

theorem *TBtheorem4a-empty*:
assumes $(ine\ P\ E) \vee (ine\ Q\ E)$
and $subcomponents\ PQ = \{P,Q\}$
and *correctCompositionIn PQ*
and $loc\ PQ = \{\}$
shows $ine\ PQ\ E$
⟨*proof*⟩

theorem *TBtheorem4a-P*:
assumes $ine\ P\ E$
and $subcomponents\ PQ = \{P,Q\}$
and *correctCompositionIn PQ*
and $\exists\ ch.\ (ch \in (ins\ P) \wedge exprChannel\ ch\ E \wedge ch \notin (loc\ PQ))$
shows $ine\ PQ\ E$
⟨*proof*⟩

theorem *TBtheorem4b-P*:
assumes $ineM\ P\ M\ E$
and $subcomponents\ PQ = \{P,Q\}$
and *correctCompositionIn PQ*
and $\exists\ ch.\ ((ch \in (ins\ Q)) \wedge (exprChannel\ ch\ E) \wedge$
$(ch \notin (loc\ PQ)) \wedge (ch \in M))$
shows $ineM\ PQ\ M\ E$
⟨*proof*⟩

theorem *TBtheorem4a-PQ*:
assumes $(ine\ P\ E) \vee (ine\ Q\ E)$
and $subcomponents\ PQ = \{P,Q\}$
and *correctCompositionIn PQ*
and $\exists\ ch.\ (((ch \in (ins\ P)) \vee (ch \in (ins\ Q))) \wedge$
$(exprChannel\ ch\ E) \wedge\ (ch \notin (loc\ PQ)))$
shows $ine\ PQ\ E$
⟨*proof*⟩

theorem *TBtheorem4b-PQ*:

assumes $(ineM\ P\ M\ E) \vee (ineM\ Q\ M\ E)$
and $subcomponents\ PQ = \{P,Q\}$
and $correctCompositionIn\ PQ$
and $\exists\ ch.\ (((ch \in (ins\ P)) \vee (ch \in (ins\ Q))) \wedge$
$(ch \in M) \wedge (exprChannel\ ch\ E) \wedge\ (ch \notin (loc\ PQ)))$
shows $ineM\ PQ\ M\ E$
⟨*proof*⟩

theorem *TBtheorem4a-notP1*:
assumes $ine\ P\ E$
and $\neg\ ine\ Q\ E$
and $subcomponents\ PQ = \{P,Q\}$
and $correctCompositionIn\ PQ$
and $\exists\ ch.\ ((ine\text{-}exprChannelSingle\ P\ ch\ E) \wedge (ch \in (loc\ PQ)))$
shows $\neg\ ine\ PQ\ E$
⟨*proof*⟩

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

theorem *TBtheorem4a-notP2*:
assumes $\neg\ ine\ Q\ E$
and $subcomponents\ PQ = \{P,Q\}$
and $correctCompositionIn\ PQ$
and $ine\text{-}exprChannelSet\ P\ ChSet\ E$
and $\forall\ (x :: chanID).\ ((x \in ChSet) \longrightarrow (x \in (loc\ PQ)))$
shows $\neg\ ine\ PQ\ E$
⟨*proof*⟩

theorem *TBtheorem4b-notP2*:
assumes $\neg\ ineM\ Q\ M\ E$
and $subcomponents\ PQ = \{P,Q\}$
and $correctCompositionIn\ PQ$
and $ine\text{-}exprChannelSet\ P\ ChSet\ E$
and $\forall\ (x :: chanID).\ ((x \in ChSet) \longrightarrow (x \in (loc\ PQ)))$
shows $\neg\ ineM\ PQ\ M\ E$
⟨*proof*⟩

theorem *TBtheorem4a-notPQ*:
assumes $subcomponents\ PQ = \{P,Q\}$
and $correctCompositionIn\ PQ$
and $ine\text{-}exprChannelSet\ P\ ChSetP\ E$

and *ine-exprChannelSet Q ChSetQ E*
and $\forall\ (x :: chanID).\ ((x \in ChSetP) \longrightarrow (x \in (loc\ PQ)))$
and $\forall\ (x :: chanID).\ ((x \in ChSetQ) \longrightarrow (x \in (loc\ PQ)))$
shows $\neg\ ine\ PQ\ E$
⟨*proof*⟩

lemma *ineM-Un1*:
assumes *ineM P A E*
shows *ineM P (A Un B) E*
⟨*proof*⟩

theorem *TBtheorem4b-notPQ*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *ine-exprChannelSet P ChSetP E*
and *ine-exprChannelSet Q ChSetQ E*
and $\forall\ (x :: chanID).\ ((x \in ChSetP) \longrightarrow (x \in (loc\ PQ)))$
and $\forall\ (x :: chanID).\ ((x \in ChSetQ) \longrightarrow (x \in (loc\ PQ)))$
shows $\neg\ ineM\ PQ\ M\ E$
⟨*proof*⟩

lemma *ine-nonempty-exprChannelSet*:
assumes *ine-exprChannelSet P ChSet E*
and *ChSet* ≠ {}
shows *ine P E*
⟨*proof*⟩

lemma *ine-empty-exprChannelSet*:
assumes *ine-exprChannelSet P ChSet E*
and *ChSet* = {}
shows $\neg\ ine\ P\ E$
⟨*proof*⟩

theorem *TBtheorem5a-empty*:
assumes $(eout\ P\ E) \vee (eout\ Q\ E)$
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *loc PQ* = {}
shows *eout PQ E*
⟨*proof*⟩

theorem *TBtheorem45a-P*:
assumes *eout P E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and $\exists\ ch.\ ((ch \in (out\ P)) \wedge (exprChannel\ ch\ E) \wedge (ch \notin (loc\ PQ)))$
shows *eout PQ E*
⟨*proof*⟩

theorem *TBtheore54b-P*:
assumes *eoutM P M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and ∃ *ch*. ((*ch* ∈ (*out Q*)) ∧ (*exprChannel ch E*) ∧
(*ch* ∉ (*loc PQ*)) ∧ (*ch* ∈ *M*))
shows *eoutM PQ M E*
⟨*proof*⟩

theorem *TBtheorem5a-PQ*:
assumes (*eout P E*) ∨ (*eout Q E*)
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and ∃ *ch*. (((*ch* ∈ (*out P*)) ∨ (*ch* ∈ (*out Q*))) ∧
(*exprChannel ch E*) ∧ (*ch* ∉ (*loc PQ*)))
shows *eout PQ E*
⟨*proof*⟩

theorem *TBtheorem5b-PQ*:
assumes (*eoutM P M E*) ∨ (*eoutM Q M E*)
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and ∃ *ch*. (((*ch* ∈ (*out P*)) ∨ (*ch* ∈ (*out Q*))) ∧ (*ch* ∈ *M*)
∧ (*exprChannel ch E*) ∧ (*ch* ∉ (*loc PQ*)))
shows *eoutM PQ M E*
⟨*proof*⟩

theorem *TBtheorem5a-notP1*:
assumes *eout P E*
and ¬ *eout Q E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and ∃ *ch*. ((*out-exprChannelSingle P ch E*) ∧ (*ch* ∈ (*loc PQ*)))
shows ¬ *eout PQ E*
⟨*proof*⟩

theorem *TBtheorem5b-notP1*:
assumes *eoutM P M E*
and ¬ *eoutM Q M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and ∃ *ch*. ((*out-exprChannelSingle P ch E*) ∧ (*ch* ∈ *M*)
∧ (*ch* ∈ (*loc PQ*)))
shows ¬ *eoutM PQ M E*
⟨*proof*⟩

theorem *TBtheorem5a-notP2*:
assumes ¬ *eout Q E*

and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *out-exprChannelSet P ChSet E*
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSet*) ⟶ (*x* ∈ (*loc PQ*)))
shows ¬ *eout PQ E*
⟨*proof*⟩

theorem *TBtheorem5b-notP2*:
assumes ¬ *eoutM Q M E*
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *out-exprChannelSet P ChSet E*
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSet*) ⟶ (*x* ∈ (*loc PQ*)))
shows ¬ *eoutM PQ M E*
⟨*proof*⟩

theorem *TBtheorem5a-notPQ*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *out-exprChannelSet P ChSetP E*
and *out-exprChannelSet Q ChSetQ E*
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSetP*) ⟶ (*x* ∈ (*loc PQ*)))
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSetQ*) ⟶ (*x* ∈ (*loc PQ*)))
shows ¬ *eout PQ E*
⟨*proof*⟩

theorem *TBtheorem5b-notPQ*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionOut PQ*
and *out-exprChannelSet P ChSetP E*
and *out-exprChannelSet Q ChSetQ E*
and *M* = *ChSetP* ∪ *ChSetQ*
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSetP*) ⟶ (*x* ∈ (*loc PQ*)))
and ∀ (*x* ::*chanID*). ((*x* ∈ *ChSetQ*) ⟶ (*x* ∈ (*loc PQ*)))
shows ¬ *eoutM PQ M E*
⟨*proof*⟩

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

lemma *LocalSecretsComposition1*:
assumes $ls \in LocalSecrets\ P$
and $subcomponents\ PQ = \{P,\ Q\}$
shows $ls \in LocalSecrets\ PQ$
⟨*proof*⟩

lemma *LocalSecretsComposition-exprChannel-k*:
assumes $exprChannel\ x\ (kE\ Keys)$
and $\neg\ ine\ P\ (kE\ Keys)$
and $\neg\ ine\ Q\ (kE\ Keys)$
and $\neg\ (x \notin ins\ P \wedge x \notin ins\ Q)$
shows *False*
⟨*proof*⟩

lemma *LocalSecretsComposition-exprChannel-s*:
assumes $exprChannel\ x\ (sE\ Secrets)$
and $\neg\ ine\ P\ (sE\ Secrets)$
and $\neg\ ine\ Q\ (sE\ Secrets)$
and $\neg\ (x \notin ins\ P \wedge x \notin ins\ Q)$
shows *False*
⟨*proof*⟩

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 $kKS\ Keys \notin LocalSecrets\ Q$
shows $kKS\ Keys \notin LocalSecrets\ PQ$
⟨*proof*⟩

lemma *LocalSecretsComposition-neg-k*:
assumes $subcomponents\ PQ = \{P,Q\}$
and $correctCompositionLoc\ PQ$
and $correctCompositionKS\ PQ$
and $(kKS\ m) \notin specKeysSecrets\ P$
and $(kKS\ m) \notin specKeysSecrets\ Q$
and $\neg\ ine\ P\ (kE\ m)$
and $\neg\ ine\ Q\ (kE\ m)$

and $(kKS\ m) \notin ((LocalSecrets\ P) \cup (LocalSecrets\ Q))$
shows $(kKS\ m) \notin (LocalSecrets\ PQ)$
⟨*proof*⟩

lemma *LocalSecretsComposition-neg-s*:
assumes *subPQ*:$subcomponents\ PQ = \{P,Q\}$
and *cCompLoc*:$correctCompositionLoc\ PQ$
and *cCompKS*:$correctCompositionKS\ PQ$
and *notKSP*:$(sKS\ m) \notin specKeysSecrets\ P$
and *notKSQ*:$(sKS\ m) \notin specKeysSecrets\ Q$
and $\neg\ ine\ P\ (sE\ m)$
and $\neg\ ine\ Q\ (sE\ m)$
and *notLocSeqPQ*:$(sKS\ m) \notin ((LocalSecrets\ P) \cup (LocalSecrets\ Q))$
shows $(sKS\ m) \notin (LocalSecrets\ PQ)$
⟨*proof*⟩

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

lemma *LocalSecretsComposition-neg1-s*:
assumes $subcomponents\ PQ = \{P,\ Q\}$
and $correctCompositionLoc\ PQ$
and $\neg\ ine\ P\ (sE\ s)$
and $\neg\ ine\ Q\ (sE\ s)$
and $sKS\ s \notin LocalSecrets\ P$
and $sKS\ s \notin LocalSecrets\ Q$
shows $sKS\ s \notin LocalSecrets\ PQ$
⟨*proof*⟩

lemma *LocalSecretsComposition-neg1*:
assumes $subcomponents\ PQ = \{P,\ Q\}$
and $correctCompositionLoc\ PQ$
and *h1*:$\forall\ m.\ ks = kKS\ m \longrightarrow (\neg\ ine\ P\ (kE\ m) \wedge \neg\ ine\ Q\ (kE\ m))$
and *h2*:$\forall\ m.\ ks = sKS\ m \longrightarrow (\neg\ ine\ P\ (sE\ m) \wedge \neg\ ine\ Q\ (sE\ m))$
and $ks \notin LocalSecrets\ P$
and $ks \notin LocalSecrets\ Q$
shows $ks \notin LocalSecrets\ PQ$
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *LocalSecretsComposition-neg-loc-k*:
assumes *kKS key* ∉ *LocalSecrets P*
and *exprChannel ch* (*kE key*)
and *kKS key* ∉ *specKeysSecrets P*
shows *ch* ∉ *loc P*
⟨*proof*⟩

lemma *LocalSecretsComposition-neg-loc-s*:
assumes *sKS secret* ∉ *LocalSecrets P*
and *exprChannel ch* (*sE secret*)

and *sKS secret* $\notin$ *specKeysSecrets P*
shows *ch* $\notin$ *loc P*
⟨*proof*⟩

lemma *correctCompositionKS-exprChannel-k-P*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *kKS key* $\notin$ *LocalSecrets PQ*
and *ch* $\in$ *ins P*
and *exprChannel ch* (*kE key*)
and *kKS key* $\notin$ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows *ch* $\in$ *ins PQ* $\wedge$ *exprChannel ch* (*kE key*)
⟨*proof*⟩

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

lemma *correctCompositionKS-exprChannel-k-Q*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *kKS key* $\notin$ *LocalSecrets PQ*
and *ch* $\in$ *ins Q*
and *h1*:*exprChannel ch* (*kE key*)
and *kKS key* $\notin$ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows *ch* $\in$ *ins PQ* $\wedge$ *exprChannel ch* (*kE key*)
⟨*proof*⟩

lemma *correctCompositionKS-exprChannel-k-Qex*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *kKS key* $\notin$ *LocalSecrets PQ*
and *ch* $\in$ *ins Q*
and *exprChannel ch* (*kE key*)
and *kKS key* $\notin$ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows $\exists$ *ch*. *ch* $\in$ *ins PQ* $\wedge$ *exprChannel ch* (*kE key*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *correctCompositionKS-exprChannel-s-Q*:
assumes *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionKS PQ*
and *sKS secret* ∉ *LocalSecrets PQ*
and *ch* ∈ *ins Q*
and *h1*:*exprChannel ch* (*sE secret*)
and *sKS secret* ∉ *specKeysSecrets PQ*
and *correctCompositionIn PQ*
shows *ch* ∈ *ins PQ* ∧ *exprChannel ch* (*sE secret*)
⟨*proof*⟩

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

end

5 Knowledge of Keys and Secrets

theory *KnowledgeKeysSecrets*
imports *CompLocalSecrets*

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

- *A may eventually get the secret m,*
- *m belongs to the set LS_A of its local secrets,*
- *A knows some list of expressions m_2 which is an concatenations of m and some list of expressions m_1,*
- *m is a concatenation of some lists of secrets m_1 and m_2, and A knows both these secrets,*
- *A knows some secret key k^{-1} and the result of the encryption of the m with the corresponding public key,*
- *A knows some public key k and the result of the signature creation of the m with the corresponding private key,*
- *m is an encryption of some secret m_1 with a public key k, and A knows both m_1 and k,*
- *m is the result of the signature creation of the m_1 with the key k, and A knows both m_1 and k.*

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

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

$(IncrDecrKeys\ k1\ k2) \wedge (know\ A\ (kKS\ k1)) \wedge (knows\ A\ (Sign\ k2\ e))$
$\longrightarrow knows\ A\ e$
and
knows6:
$(know\ A\ (kKS\ k)) \wedge (knows\ A\ e1) \longrightarrow knows\ A\ (Enc\ k\ e1)$
and
knows7:
$(know\ A\ (kKS\ k)) \wedge (knows\ A\ e1) \longrightarrow knows\ A\ (Sign\ k\ e1)$

primrec $eoutKnowCorrect :: specID \Rightarrow KS \Rightarrow bool$
where
eout-know-k:
$eoutKnowCorrect\ C\ (kKS\ m) =$
$((eout\ \ C\ (kE\ m)) \longleftrightarrow (m \in (specKeys\ C) \vee (know\ C\ (kKS\ m)))\ )$ |
eout-know-s:
$eoutKnowCorrect\ C\ (sKS\ m) =$
$((eout\ \ C\ (sE\ m)) \longleftrightarrow\ (m \in (specSecrets\ C) \vee (know\ C\ (sKS\ m)))\ )$

definition $eoutKnowsECorrect :: specID \Rightarrow Expression \Rightarrow bool$
where
$eoutKnowsECorrect\ C\ e \equiv$
$((eout\ \ C\ e) \longleftrightarrow$
$((\exists\ k.\ e = (kE\ k) \wedge (k \in specKeys\ C)) \vee$
$(\exists\ s.\ e = (sE\ s) \wedge (s \in specSecrets\ C)) \vee$
$(knows\ C\ [e])))$

lemma *eoutKnowCorrect-L1k*:
assumes $eoutKnowCorrect\ C\ (kKS\ m)$
and $eout\ \ C\ (kE\ m)$
shows $m \in (specKeys\ C) \vee (know\ C\ (kKS\ m))$
⟨*proof*⟩

lemma *eoutKnowCorrect-L1s*:
assumes $eoutKnowCorrect\ C\ (sKS\ m)$
and $eout\ \ C\ (sE\ m)$
shows $m \in (specSecrets\ C) \vee (know\ C\ (sKS\ m))$
⟨*proof*⟩

lemma *eoutKnowsECorrect-L1*:
assumes $eoutKnowsECorrect\ C\ e$
and $eout\ \ C\ e$
shows $(\exists\ k.\ e = (kE\ k) \wedge (k \in specKeys\ C)) \vee$
$(\exists\ s.\ e = (sE\ s) \wedge (s \in specSecrets\ C)) \vee$
$(knows\ C\ [e])$
⟨*proof*⟩

lemma *know2knows-k*:
assumes $know\ A\ (kKS\ m)$
shows $knows\ A\ [kE\ m]$

⟨*proof*⟩

lemma *knows2know-k*:
assumes *knows A* [*kE m*]
shows *know A* (*kKS m*)
⟨*proof*⟩

lemma *know2knowsPQ-k*:
assumes *know P* (*kKS m*) ∨ *know Q* (*kKS m*)
shows *knows P* [*kE m*] ∨ *knows Q* [*kE m*]
⟨*proof*⟩

lemma *knows2knowPQ-k*:
assumes *knows P* [*kE m*] ∨ *knows Q* [*kE m*]
shows *know P* (*kKS m*) ∨ *know Q* (*kKS m*)
⟨*proof*⟩

lemma *knows1k*:
know A (*kKS m*) = *knows A* [*kE m*]
⟨*proof*⟩

lemma *know2knows-neg-k*:
assumes ¬ *know A* (*kKS m*)
shows ¬ *knows A* [*kE m*]
⟨*proof*⟩

lemma *knows2know-neg-k*:
assumes ¬ *knows A* [*kE m*]
shows ¬ *know A* (*kKS m*)
⟨*proof*⟩

lemma *know2knows-s*:
assumes *know A* (*sKS m*)
shows *knows A* [*sE m*]
⟨*proof*⟩

lemma *knows2know-s*:
assumes *knows A* [*sE m*]
shows *know A* (*sKS m*)
⟨*proof*⟩

lemma *know2knowsPQ-s*:
assumes *know P* (*sKS m*) ∨ *know Q* (*sKS m*)
shows *knows P* [*sE m*] ∨ *knows Q* [*sE m*]
⟨*proof*⟩

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

⟨*proof*⟩

lemma *knows1s*:
know A (*sKS m*) = *knows A* [*sE m*]
⟨*proof*⟩

lemma *know2knows-neg-s*:
assumes ¬ *know A* (*sKS m*)
shows ¬ *knows A* [*sE m*]
⟨*proof*⟩

lemma *knows2know-neg-s*:
assumes ¬ *knows A* [*sE m*]
shows ¬ *know A* (*sKS m*)
⟨*proof*⟩

lemma *knows2*:
assumes *e2* = *e1* @ *e* ∨ *e2* = *e* @ *e1*
and *knows A e2*
shows *knows A e*
⟨*proof*⟩

lemma *correctCompositionInLoc-exprChannel*:
assumes *subcomponents PQ* = {*P*, *Q*}
and *correctCompositionIn PQ*
and *ch* : *ins P*
and *exprChannel ch m*
and ∀ *x*. *x* ∈ *ins PQ* ⟶ ¬ *exprChannel x m*
shows *ch* : *loc PQ*
⟨*proof*⟩

lemma *eout-know-nonKS-k*:
assumes *m* ∉ *specKeys A*
and *eout A* (*kE m*)
and *eoutKnowCorrect A* (*kKS m*)
shows *know A* (*kKS m*)
⟨*proof*⟩

lemma *eout-know-nonKS-s*:
assumes *m* ∉ *specSecrets A*
and *eout A* (*sE m*)
and *eoutKnowCorrect A* (*sKS m*)
shows *know A* (*sKS m*)
⟨*proof*⟩

lemma *not-know-k-not-ine*:
assumes ¬ *know A* (*kKS m*)
shows ¬ *ine A* (*kE m*)
⟨*proof*⟩

lemma *not-know-s-not-ine*:
assumes ¬ *know A* (*sKS m*)
shows ¬ *ine A* (*sE m*)
⟨*proof*⟩

lemma *not-know-k-not-eout*:
assumes *m* ∉ *specKeys A*
and ¬ *know A* (*kKS m*)
and *eoutKnowCorrect A* (*kKS m*)
shows ¬ *eout A* (*kE m*)
⟨*proof*⟩

lemma *not-know-s-not-eout*:
assumes *m* ∉ *specSecrets A*
and ¬ *know A* (*sKS m*)
and *eoutKnowCorrect A* (*sKS m*)
shows ¬ *eout A* (*sE m*)
⟨*proof*⟩

lemma *adv-not-know1*:
assumes *out P* ⊆ *ins A*
and ¬ *know A* (*kKS m*)
shows ¬ *eout P* (*kE m*)
⟨*proof*⟩

lemma *adv-not-know2*:
assumes *out P* ⊆ *ins A*
and ¬ *know A* (*sKS m*)
shows ¬ *eout P* (*sE m*)
⟨*proof*⟩

lemma *LocalSecrets-L1*:
assumes (*kKS*) *key* ∈ *LocalSecrets P*
and (*kKS key*) ∉ ⋃(*LocalSecrets* ‘ *subcomponents P*)
shows *kKS key* ∉ *specKeysSecrets P*
⟨*proof*⟩

lemma *LocalSecrets-L2*:
assumes *kKS key* ∈ *LocalSecrets P*
and *kKS key* ∈ *specKeysSecrets P*
shows *kKS key* ∈ ⋃(*LocalSecrets* ‘ *subcomponents P*)
⟨*proof*⟩

lemma *know-composition1*:
assumes *notKSP*:*m* ∉ *specKeysSecrets P*
and *notKSQ*:*m* ∉ *specKeysSecrets Q*
and *know P m*
and *subPQ*:*subcomponents PQ* = {*P*,*Q*}

and *cCompI*:*correctCompositionIn PQ*
and *cCompKS*:*correctCompositionKS PQ*
shows *know PQ m*
⟨*proof*⟩

lemma *know-composition2*:
assumes $m \notin$ *specKeysSecrets P*
and $m \notin$ *specKeysSecrets Q*
and *know Q m*
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *know PQ m*
⟨*proof*⟩

lemma *know-composition*:
assumes $m \notin$ *specKeysSecrets P*
and $m \notin$ *specKeysSecrets Q*
and *know P m* $\vee$ *know Q m*
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *know PQ m*
⟨*proof*⟩

theorem *know-composition-neg-ine-k*:
assumes $\neg$ *know P* (*kKS key*)
and $\neg$ *know Q* (*kKS key*)
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
shows $\neg$ (*ine PQ* (*kE key*))
⟨*proof*⟩

theorem *know-composition-neg-ine-s*:
assumes $\neg$ *know P* (*sKS secret*)
and $\neg$ *know Q* (*sKS secret*)
and *subcomponents PQ* = $\{P,Q\}$
and *correctCompositionIn PQ*
shows $\neg$ (*ine PQ* (*sE secret*))
⟨*proof*⟩

lemma *know-composition-neg1*:
assumes *notknowP*:$\neg$ *know P m*
and *notknowQ*:$\neg$ *know Q m*
and *subPQ*:*subcomponents PQ* = $\{P,Q\}$
and *cCompLoc*:*correctCompositionLoc PQ*
and *cCompI*:*correctCompositionIn PQ*
shows $\neg$ *know PQ m*
⟨*proof*⟩

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* $\vee$ *know Q m*
⟨*proof*⟩

lemma *eout-knows-nonKS-k*:
assumes *m* $\notin$ (*specKeys A*)
and *eout A* (*kE m*)
and *eoutKnowsECorrect A* (*kE m*)
shows *knows A* [*kE m*]
⟨*proof*⟩

lemma *eout-knows-nonKS-s*:
assumes *h1*:*m* $\notin$ *specSecrets A*
and *h2*:*eout A* (*sE m*)
and *h3*:*eoutKnowsECorrect A* (*sE m*)
shows *knows A* [*sE m*]
⟨*proof*⟩

lemma *not-knows-k-not-ine*:
assumes $\neg$ *knows A* [*kE m*]
shows $\neg$ *ine A* (*kE m*)
⟨*proof*⟩

lemma *not-knows-s-not-ine*:
assumes $\neg$ *knows A* [*sE m*]
shows $\neg$ *ine A* (*sE m*)
⟨*proof*⟩

lemma *not-knows-k-not-eout*:
assumes *m* $\notin$ *specKeys A*
and $\neg$ *knows A* [*kE m*]
and *eoutKnowsECorrect A* (*kE m*)
shows $\neg$ *eout A* (*kE m*)
⟨*proof*⟩

lemma *not-knows-s-not-eout*:
assumes *m* $\notin$ *specSecrets A*
and $\neg$ *knows A* [*sE m*]
and *eoutKnowsECorrect A* (*sE m*)
shows $\neg$ *eout A* (*sE m*)
⟨*proof*⟩

lemma *adv-not-knows1*:
assumes *out P* $\subseteq$ *ins A*

and ¬ *knows A* [*kE m*]
shows ¬ *eout P* (*kE m*)
⟨*proof*⟩

lemma *adv-not-knows2*:
assumes *out P* ⊆ *ins A*
and ¬ *knows A* [*sE m*]
shows ¬ *eout P* (*sE m*)
⟨*proof*⟩

lemma *knows-decomposition-1-k*:
assumes *kKS a* ∉ *specKeysSecrets P*
and *kKS a* ∉ *specKeysSecrets Q*
and *subcomponents PQ* = {*P*, *Q*}
and *knows PQ* [*kE a*]
and *correctCompositionIn PQ*
and *correctCompositionLoc PQ*
shows *knows P* [*kE a*] ∨ *knows Q* [*kE a*]
⟨*proof*⟩

lemma *knows-decomposition-1-s*:
assumes *sKS a* ∉ *specKeysSecrets P*
and *sKS a* ∉ *specKeysSecrets Q*
and *subcomponents PQ* = {*P*, *Q*}
and *knows PQ* [*sE a*]
and *correctCompositionIn PQ*
and *correctCompositionLoc PQ*
shows *knows P* [*sE a*] ∨ *knows Q* [*sE a*]
⟨*proof*⟩

lemma *knows-decomposition-1*:
assumes *subcomponents PQ* = {*P*, *Q*}
and *knows PQ* [*a*]
and *correctCompositionIn PQ*
and *correctCompositionLoc PQ*
and (∃ *z*. *a* = *kE z*) ∨ (∃ *z*. *a* = *sE z*)
and ∀ *z*. *a* = *kE z* ⟶
kKS z ∉ *specKeysSecrets P* ∧ *kKS z* ∉ *specKeysSecrets Q*
and *h7*:∀ *z*. *a* = *sE z* ⟶
sKS z ∉ *specKeysSecrets P* ∧ *sKS z* ∉ *specKeysSecrets Q*
shows *knows P* [*a*] ∨ *knows Q* [*a*]
⟨*proof*⟩

lemma *knows-composition1-k*:
assumes (*kKS m*) ∉ *specKeysSecrets P*
and (*kKS m*) ∉ *specKeysSecrets Q*
and *knows P* [*kE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*

and *correctCompositionKS PQ*
shows *knows PQ* [*kE m*]
⟨*proof*⟩

lemma *knows-composition1-s*:
assumes (*sKS m*) ∉ *specKeysSecrets P*
and (*sKS m*) ∉ *specKeysSecrets Q*
and *knows P* [*sE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *knows PQ* [*sE m*]
⟨*proof*⟩

lemma *knows-composition2-k*:
assumes (*kKS m*) ∉ *specKeysSecrets P*
and (*kKS m*) ∉ *specKeysSecrets Q*
and *knows Q* [*kE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *knows PQ* [*kE m*]
⟨*proof*⟩

lemma *knows-composition2-s*:
assumes (*sKS m*) ∉ *specKeysSecrets P*
and (*sKS m*) ∉ *specKeysSecrets Q*
and *knows Q* [*sE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows *knows PQ* [*sE m*]
⟨*proof*⟩

lemma *knows-composition-neg1-k*:
assumes *kKS m* ∉ *specKeysSecrets P*
and *kKS m* ∉ *specKeysSecrets Q*
and ¬ *knows P* [*kE m*]
and ¬ *knows Q* [*kE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionLoc PQ*
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows ¬ *knows PQ* [*kE m*]
⟨*proof*⟩

lemma *knows-composition-neg1-s*:
assumes *sKS m* ∉ *specKeysSecrets P*
and *sKS m* ∉ *specKeysSecrets Q*

and ¬ *knows P* [*sE m*]
and ¬ *knows Q* [*sE m*]
and *subcomponents PQ* = {*P*,*Q*}
and *correctCompositionLoc PQ*
and *correctCompositionIn PQ*
and *correctCompositionKS PQ*
shows ¬ *knows PQ* [*sE m*]
⟨*proof*⟩

lemma *knows-concat-1*:
assumes *knows P* (*a* # *e*)
shows *knows P* [*a*]
⟨*proof*⟩

lemma *knows-concat-2*:
assumes *knows P* (*a* # *e*)
shows *knows P e*
⟨*proof*⟩

lemma *knows-concat-3*:
assumes *knows P* [*a*]
and *knows P e*
shows *knows P* (*a* # *e*)
⟨*proof*⟩

lemma *not-knows-conc-knows-elem-not-knows-tail*:
assumes ¬ *knows P* (*a* # *e*)
and *knows P* [*a*]
shows ¬ *knows P e*
⟨*proof*⟩

lemma *not-knows-conc-not-knows-elem-tail*:
assumes ¬ *knows P* (*a*#*e*)
shows ¬ *knows P* [*a*] ∨ ¬ *knows P e*
⟨*proof*⟩

lemma *not-knows-elem-not-knows-conc*:
assumes ¬ *knows P* [*a*]
shows ¬ *knows P* (*a* # *e*)
⟨*proof*⟩

lemma *not-knows-tail-not-knows-conc*:
assumes ¬ *knows P e*
shows ¬ *knows P* (*a* # *e*)
⟨*proof*⟩

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*
⟨*proof*⟩

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

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

end

References

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

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

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