Logging-independent Message Anonymity in the Relational Method

Pasquale Noce
Software Engineer at HID Global, Italy
pasquale dot noce dot lavoro at gmail dot com
pasquale dot noce at hidglobal dot com

May 26, 2024

Abstract

In the context of formal cryptographic protocol verification, logging-independent message anonymity is the property for a given message to remain anonymous despite the attacker's capability of mapping messages of that sort to agents based on some intrinsic feature of such messages, rather than by logging the messages exchanged by legitimate agents as with logging-dependent message anonymity.

This paper illustrates how logging-independent message anonymity can be formalized according to the relational method for formal protocol verification by considering a real-world protocol, namely the Restricted Identification one by the BSI. This sample model is used to verify that the pseudonymous identifiers output by user identification tokens remain anonymous under the expected conditions.

Contents

1 Logging-independent message anonymity and Restricted Identification **2**
1.1 Introduction 2
1.2 Case study: the Restricted Identification protocol 2
1.3 Agents, messages, protocol rules 5

2 Anonymity of token pseudonymous identifiers **11**

3 Possibility of anonymity compromise for token pseudonymous identifiers **15**

1 Logging-independent message anonymity and Restricted Identification

theory *Definitions*
imports *Main*
begin

1.1 Introduction

Logging-dependent message anonymity is the property for a message exchanged or otherwise used in a cryptographic protocol to remain anonymous although the attacker can log the messages generated or accepted by legitimate agents, and map any two observable messages contained in any such message to the same agent. An approach to modeling and verifying this security property according to the *relational method* for formal protocol verification has been described in [7], along with the method itself. This approach makes use of two further type constructors for messages, *IDInfo* and *Log*, as well as of two functions, *crypts* and *key-sets*. Particularly, *IDInfo* is used to model message anonymity, while the remaining constants are used to formalize the property for a message to be observable by the spy within some logged message.

Logging-independent message anonymity rather is the property for a given message to remain anonymous in spite of the attacker's capability of mapping messages of that sort to agents without resorting to message logging, namely by means of some intrinsic feature of such messages. From the above observation, it follows that *IDInfo* is the sole anonymity-related constant required if the only kind of anonymity of interest for a given protocol is the logging-independent one, whereas *Log*, *crypts*, and *key-sets* are unnecessary and can be left out of the model. It is also possible to include both kinds of anonymity in the model, in which case some protocol rule will enable the spy to map messages of some sort to agents only if they are observable within logged messages, while some other protocol rule will enable the spy to do so independently of this condition.

This paper illustrates how logging-independent message anonymity can be formalized according to the relational method by considering a real-world protocol, namely the Restricted Identification one by the BSI [1] [2], whose very purpose is to allow for the exchange of messages endowed with this security property.

1.2 Case study: the Restricted Identification protocol

The Restricted Identification protocol enables *user identification tokens* (e.g. electronic documents) to generate and output unambiguous *pseudonymous identifiers*, distinct for any given group of terminals of arbitrary granularity,

referred to as a *sector*, and usable to identify the tokens across different sessions taking place within the same sector. For example, such identifiers allow for the creation of sector-specific revocation lists, at the same time preserving the anonymity of the holder of any token included in such a list.

This protocol is based on a *Public Key Infrastructure (PKI)* comprising the token issuer, which owns a *revocation key pair* (SK_{Rev}, PK_{Rev}) and generates a *token key pair* (SK_{Tok}, PK_{Tok}) for each token, and sectors, each one endowed with its own *sector key pair* (SK_{Sec}, PK_{Sec}), where $PK_{Sec} = [SK_{Sec}]PK_{Rev}$. This PKI may use either an integer finite field or an elliptic curve group, as long as the selected domain parameters are cryptographically secure. In a real-world PKI, each sector has actually two distinct key pairs, which enables the tokens to generate as many different pseudonymous identifiers per sector, but this detail is irrelevant to the anonymity of such identifiers and can then be omitted from the model.

According to [1] [2], the Restricted Identification protocol may only be executed using the session keys established via Chip Authentication version 2/3, after performing Terminal Authentication version 2 with a terminal certificate containing both sector public keys' hashes (including domain parameters), whose authenticity is ensured by the certificate's signature. After requesting to start the protocol, which again is irrelevant and can be left out of the model, the terminal sends either of its sector public keys PK_{Sec} (including domain parameters) to the token, which in turn verifies that PK_{Sec}'s hash matches the one contained in the certificate and replies with its pseudonymous identifier $H([SK_{Tok}]PK_{Sec})$, where H is a hash function. If necessary (for instance, to insert it into a sector-specific revocation list), this identifier can be recomputed in the external world as $H([SK_{Sec}]([SK_{Rev}]PK_{Tok}))$, with the concurrence of both the token issuer and the entity responsible for the involved sector.

As a matter of fact, since the only purpose of the protocol model to be developed is to verify the logging-independent anonymity of token pseudonymous identifiers, without any confidentiality or authenticity concern, it is sufficient to model a simpler protocol in which the terminal and the token exchange their messages in plain, without using any session key. Moreover, since both Chip and Terminal Authentication are out of scope, the Restricted Identification protocol will be modeled as a stand-alone one. Consequently, although both of them are exchanged during Terminal Authentication in the real-world protocol, PK_{Sec}'s signature will be assumed to be exchanged with PK_{Sec} in the model, while the related verification key will be assumed to be known by the token a priori. The hash function used to sign PK_{Sec} may differ from the one used to compute token pseudonymous identifiers, but once more, this is just an omissible functional detail.

A further simplification, admissible for the same reason, is to let the token use domain parameters known a priori rather than the input ones, whose

presence in the input message can then be left out. Indeed, this prevents the spy from snatching SK_{Tok} by making an element of a smaller group pass for an authentic sector public key, which could be done by signing it with a compromised signature generation key. For example, if the PKI used a group of 128-bit order, SK_{Tok} could be disclosed by first searching the private key SK'_{Tok} associated with a fake identifier ranging in a group of 64-bit order n, and then detecting SK_{Tok} as the unique private key associated with a given genuine identifier among all those differing from SK'_{Tok} by a multiple of n. So, two searches within as many spaces of 2^{64} elements, which is a computationally feasible task nowadays, would suffice to find SK_{Tok}. However, such *small group attacks* can safely be ruled out as long as the initial state s_0 comprises an arbitrary set of compromised token private keys, given that verifying the conditions under which these keys remain secret is out of scope.

As a result of all the simplifications described above, the protocol that is going to be modeled is as follows.

1. Terminal $\rightarrow$ Token: $\{PK_{Sec}, \{H(PK_{Sec})\}_{SK_{Sign}}\}$

   The terminal sends the token a message consisting of its sector public key and a precomputed signature of this key.

2. Token $\rightarrow$ Terminal: $H([SK_{Tok}]PK_{Sec})$

   The token verifies that the hash of the received public key matches the signed one, and then replies with the pseudonymous identifier resulting from this key.

Unless it is compromised by means other than attacking the protocol, the anonymity of a given token pseudonymous identifier $H([SK_{Tok}]PK_{Sec})$ is expected to be vulnerable if and only if either SK_{Tok} is anonymity-compromised, or SK_{Sec} is compromised and there is another compromised sector private key SK'_{Sec} such that $H([SK_{Tok}]PK'_{Sec})$ is anonymity-compromised. In fact, the spy can detect the use of SK_{Tok} in $H([SK_{Tok}]PK_{Sec})$ in the former case, whereas in the latter one he can map $H([SK_{Tok}]PK_{Sec})$ to the same token as $H([SK_{Tok}]PK'_{Sec})$ by recognizing that $[SK_{Tok}]PK_{Sec} = [SK_{Sec} \times (SK'_{Sec})^{-1}]([SK_{Tok}]PK'_{Sec})$.

The purpose of the following formal development is precisely to formally prove the correctness of this expectation. In more detail, the *only if* conditional implied by the previous statement will be proven as an *anonymity property* in the next section, while the *if* conditional will be proven in the form of two *possibility properties* in the subsequent one. Since both relevant

attack options leverage the intrinsic features of token pseudonymous identifiers, namely the private keys used to generate them, logging-independent message anonymity has to be considered rather than logging-dependent one.

For further information about the formal definitions and proofs contained in this paper, see Isabelle documentation, particularly [6], [5], [3], and [4].

1.3 Agents, messages, protocol rules

Agents consist of an infinite population of tokens and sectors, identified through natural numbers, plus the spy. Actually, the model can safely ignore terminals altogether and assume that tokens are presented to sectors as a whole, since all the terminal-side messages used in the protocol refer to sectors rather than to individual terminals. The only possible exceptions are signature key pairs. In fact, although the entity signing terminal certificates is the same for every terminal in a given sector (in [1] and [2], it is named *Document Verifier*), nothing prevents that entity to use distinct signature generation keys for different terminals (for example, if their certificates are issued at different times). Nonetheless, the granularity of signature key pairs is irrelevant to the anonymity of token pseudonymous identifiers according to the previous considerations, so these key pairs can be associated with sectors as well.

As opposed to what happens in [7], there is no correlation here between any two agents *Token* n and *Sector* n marked with the same numeric identifier n. In fact, tokens and sectors are independent of each other, and any token may be presented to whatever sector.

type-synonym *agent-id* = *nat*

datatype *agent* =
  Token agent-id |
  Sector agent-id |
  Spy

As regards the key pairs for key agreement, private keys are identified by natural numbers, whereas public keys by sets of natural numbers. The implied interpretation is that *PubK S* stands for public key $[k]G$, where G is the group generator and k is the modular product of all the private keys referred to by the numeric identifiers in S, each one occurring as a factor exactly once. Using *multisets* of natural numbers instead of sets would have allowed private keys to be used as factors even more than once, but this option can be left out as the PKI does not provide for any public key computed in this way. The need for this ad hoc message format, like those used to represent session keys and Chip Authentication Data in [7],

confirms that reuse of the spy's capabilities' model in the inductive method is hindered by the likely need for ad hoc message formats in case of protocols using nontrivial public key cryptography [7].

Besides key agreement keys, messages comprise signature generation/verification keys (identified by the numeric identifiers of the respective sectors), hash values, cryptograms, and compound messages built via message concatenation. Furthermore, message anonymity is modeled by means of constructor *IDInfo* [7]. Since the anonymity of terminal-side messages is of no concern, the interpretation of message $\langle n,\ X\rangle$ is "message X is mapped to *Token* n", namely n is always interpreted as a token's numeric identifier rather than a sector's one.

type-synonym *key-id* = *nat*

datatype *agr-key* =
  PriK key-id |
  PubK key-id set

datatype *enc-key* =
  SigK agent-id |
  VerK agent-id

datatype *msg* =
  AgrKey agr-key |
  EncKey enc-key |
  Hash msg |
  Crypt enc-key msg |
  MPair msg msg |
  IDInfo agent-id msg

syntax
  -MPair :: ['*a*, *args*] ⇒ '*a* ∗ '*b* ((2{|-,/ -|}))
  -IDInfo :: [*agent-id*, *msg*] ⇒ *msg* ((2⟨-,/ -⟩))
translations
  {|*X*, *Y*, *Z*|} ⇌ {|*X*, {|*Y*, *Z*|}|}
  {|*X*, *Y*|} ⇌ *CONST MPair X Y*
  ⟨*n*, *X*⟩ ⇌ *CONST IDInfo n X*

abbreviation *SigKey* :: *agent-id* ⇒ *msg* **where**
SigKey ≡ *EncKey* ∘ *SigK*

abbreviation *VerKey* :: *agent-id* ⇒ *msg* **where**
VerKey ≡ *EncKey* ∘ *VerK*

abbreviation *PriKey* :: *key-id* ⇒ *msg* **where**
PriKey ≡ *AgrKey* ∘ *PriK*

abbreviation *PubKey* :: *key-id set* ⇒ *msg* **where**
PubKey ≡ *AgrKey* ∘ *PubK*

primrec *InvK* :: *enc-key* ⇒ *enc-key* **where**
InvK (*SigK n*) = *VerK n* |
InvK (*VerK n*) = *SigK n*

abbreviation *InvKey* :: *enc-key* ⇒ *msg* **where**
InvKey ≡ *EncKey* ∘ *InvK*

inductive-set *parts* :: *msg set* ⇒ *msg set*
for *H* :: *msg set* **where**

parts-used [*intro*]:
X ∈ *H* ⟹ *X* ∈ *parts H* |

parts-crypt [*intro*]:
Crypt K X ∈ *parts H* ⟹ *X* ∈ *parts H* |

parts-fst [*intro*]:
{|*X*, *Y*|} ∈ *parts H* ⟹ *X* ∈ *parts H* |

parts-snd [*intro*]:
{|*X*, *Y*|} ∈ *parts H* ⟹ *Y* ∈ *parts H*

definition *parts-msg* :: *msg* ⇒ *msg set* **where**
parts-msg X ≡ *parts* {*X*}

Constant *Rev-PriK* is the numeric identifier of the revocation private key, while functions *Sec-PriK* and *Tok-PriK* map the numeric identifiers of sectors and tokens to those of the respective sector/token private keys. It is assumed that these functions are injective, as well as that their ranges do not contain *Rev-PriK* and are disjoint, and such axioms are proven to be consistent by showing that there exist three constants satisfying all of them. On the whole, these axioms just model the assumption that private keys are generated by cryptographically secure means throughout the PKI, so that the probability for any given private key to occur more than once within the PKI is negligible.

consts *Rev-PriK* :: *key-id*

consts *Sec-PriK* :: *agent-id* ⇒ *key-id*

consts *Tok-PriK* :: *agent-id* ⇒ *key-id*

specification (*Rev-PriK Sec-PriK Tok-PriK*)
  sec-prik-inj: *inj Sec-PriK*
  tok-prik-inj: *inj Tok-PriK*
  sec-prik-rev: *Rev-PriK* $\notin$ *range Sec-PriK*
  tok-prik-rev: *Rev-PriK* $\notin$ *range Tok-PriK*
  sec-prik-tok-prik: *range Sec-PriK* $\cap$ *range Tok-PriK* = {}
⟨*proof*⟩

abbreviation *Gen-PubKey* :: *msg* **where**
Gen-PubKey ≡ *PubKey* {}

abbreviation *Rev-PriKey* :: *msg* **where**
Rev-PriKey ≡ *PriKey Rev-PriK*

abbreviation *Rev-PubKey* :: *msg* **where**
Rev-PubKey ≡ *PubKey* {*Rev-PriK*}

abbreviation *Tok-PriKey* :: *agent-id* ⇒ *msg* **where**
Tok-PriKey n ≡ *PriKey* (*Tok-PriK n*)

abbreviation *Tok-PubKey* :: *agent-id* ⇒ *msg* **where**
Tok-PubKey n ≡ *PubKey* {*Tok-PriK n*}

abbreviation *Sec-PriKey* :: *agent-id* ⇒ *msg* **where**
Sec-PriKey n ≡ *PriKey* (*Sec-PriK n*)

abbreviation *Sec-PubKey* :: *agent-id* ⇒ *msg* **where**
Sec-PubKey n ≡ *PubKey* {*Sec-PriK n*, *Rev-PriK*}

abbreviation *Sign* :: *agent-id* ⇒ *msg* ⇒ *msg* **where**
Sign n X ≡ *Crypt* (*SigK n*) (*Hash X*)

abbreviation *ID* :: *agent-id* ⇒ *msg* ⇒ *msg* **where**
ID n X ≡ *case X of AgrKey* (*PubK S*) ⇒ *PubKey* (*insert* (*Tok-PriK n*) *S*)

The spy's starting knowledge, as defined by the initial state s_0, consists of the following messages.

- All the public keys used in the PKI (including the group generator).
- All token pseudonymous identifiers (in both hashed and non-hashed formats).
- Compromised private keys used in the PKI (excluding the revocation one, assumed to be secret).
- Mappings of all token public keys, compromised token private keys,

and anonymity-compromised token pseudonymous identifiers (in both hashed and non-hashed formats) to the respective tokens.

consts *bad-sigk* :: *agent-id set*

consts *bad-sec-prik* :: *agent-id set*

consts *bad-tok-prik* :: *agent-id set*

consts *bad-id* :: (*agent-id* × *agent-id*) *set*

type-synonym *event* = *agent* × *msg*

type-synonym *state* = *event set*

abbreviation *used* :: *state* ⇒ *msg set* **where**
used s ≡ *Range s*

abbreviation *spied* :: *state* ⇒ *msg set* **where**
spied s ≡ *s* `` {*Spy*}

abbreviation s_0 :: *state* **where**
s_0 ≡ {*Spy*} × ({*Gen-PubKey*, *Rev-PubKey*} ∪
SigKey ‘ *bad-sigk* ∪ *Sec-PriKey* ‘ *bad-sec-prik* ∪ *Tok-PriKey* ‘ *bad-tok-prik* ∪
range VerKey ∪ *range Sec-PubKey* ∪ *range Tok-PubKey* ∪
range (λ(*n*, *m*). *ID n* (*Sec-PubKey m*)) ∪
range (λ(*n*, *m*). *Hash* (*ID n* (*Sec-PubKey m*))) ∪
range (λ*n*. ⟨*n*, *Tok-PubKey n*⟩) ∪
{⟨*n*, *Tok-PriKey n*⟩ | *n*. *n* ∈ *bad-tok-prik*} ∪
{⟨*n*, *ID n* (*Sec-PubKey m*)⟩ | *n m*. (*n*, *m*) ∈ *bad-id*} ∪
{⟨*n*, *Hash* (*ID n* (*Sec-PubKey m*))⟩ | *n m*. (*n*, *m*) ∈ *bad-id*})

Protocol rules are defined here below. Particularly, for any public key $[SK_1 \times ... \times SK_n]G$ known to the spy, they enable him to generate public key $[SK_1 \times ... \times SK_n \times SK_{n+1}]G$ for any additional, compromised private key SK_{n+1}, as well as public key $[SK_1 \times ... \times SK_{i-1} \times SK_{i+1} \times ... \times SK_n]G$ for any compromised private key SK_i, where $1 \leq i \leq n$ (which is equivalent to multiplying the original public key by $(SK_i)^{-1}$). The spy can also map the resulting public keys to the same token, if identified, as the original public key, in the latter case as long as the related token private key still occurs as a factor in the resulting modular product. Furthermore, the spy can associate a token with any known public key whose modular product of private keys contains the corresponding token private key as a factor, provided that this key is compromised.

abbreviation *rel-sector* :: (*state* × *state*) *set* **where**
rel-sector ≡ {(*s*, *s'*) | *s s' m*.
  s' = *s* ∪ {*Sector m*, *Spy*} × {{|*Sec-PubKey m*, *Sign m* (*Sec-PubKey m*)|}}}

abbreviation *rel-token* :: (*state* × *state*) *set* **where**
rel-token ≡ {(*s*, *s'*) | *s s' m n S*.
  s' = *s* ∪ {*Token n*, *Spy*} × {*Hash* (*ID n* (*PubKey S*))} ∧
  {|*PubKey S*, *Sign m* (*PubKey S*)|} ∈ *used s*}

abbreviation *rel-pubk-less* :: (*state* × *state*) *set* **where**
rel-pubk-less ≡ {(*s*, *s'*) | *s s' A S*.
  s' = *insert* (*Spy*, *PubKey* (*S* − {*A*})) *s* ∧
  {*PriKey A*, *PubKey S*} ⊆ *spied s*}

abbreviation *rel-pubk-more* :: (*state* × *state*) *set* **where**
rel-pubk-more ≡ {(*s*, *s'*) | *s s' A S*.
  s' = *insert* (*Spy*, *PubKey* (*insert A S*)) *s* ∧
  {*PriKey A*, *PubKey S*} ⊆ *spied s*}

abbreviation *rel-hash* :: (*state* × *state*) *set* **where**
rel-hash ≡ {(*s*, *s'*) | *s s' X*.
  s' = *insert* (*Spy*, *Hash X*) *s* ∧
  X ∈ *spied s*}

abbreviation *rel-dec* :: (*state* × *state*) *set* **where**
rel-dec ≡ {(*s*, *s'*) | *s s' K X*.
  s' = *insert* (*Spy*, *X*) *s* ∧
  {*Crypt K X*, *InvKey K*} ⊆ *spied s*}

abbreviation *rel-enc* :: (*state* × *state*) *set* **where**
rel-enc ≡ {(*s*, *s'*) | *s s' K X*.
  s' = *insert* (*Spy*, *Crypt K X*) *s* ∧
  {*X*, *EncKey K*} ⊆ *spied s*}

abbreviation *rel-sep* :: (*state* × *state*) *set* **where**
rel-sep ≡ {(*s*, *s'*) | *s s' X Y*.
  s' = *s* ∪ {*Spy*} × {*X*, *Y*} ∧
  {|*X*, *Y*|} ∈ *spied s*}

abbreviation *rel-con* :: (*state* × *state*) *set* **where**
rel-con ≡ {(*s*, *s'*) | *s s' X Y*.
  s' = *insert* (*Spy*, {|*X*, *Y*|}) *s* ∧
  {*X*, *Y*} ⊆ *spied s*}

abbreviation *rel-id-pubk-less* :: (*state* × *state*) *set* **where**
rel-id-pubk-less ≡ {(*s*, *s'*) | *s s' n A S*.
  s' = *insert* (*Spy*, ⟨*n*, *PubKey* (*S* − {*A*})⟩) *s* ∧

$\{PriKey\ A,\ PubKey\ (S - \{A\}),\ \langle n,\ PubKey\ S\rangle\} \subseteq spied\ s \wedge$
$Tok\text{-}PriK\ n \in S - \{A\}\}$

abbreviation *rel-id-pubk-more* :: $(state \times state)\ set$ **where**
rel-id-pubk-more $\equiv \{(s,\ s') \mid s\ s'\ n\ A\ S.$
$s' = insert\ (Spy,\ \langle n,\ PubKey\ (insert\ A\ S)\rangle)\ s \wedge$
$\{PriKey\ A,\ PubKey\ (insert\ A\ S),\ \langle n,\ PubKey\ S\rangle\} \subseteq spied\ s\}$

abbreviation *rel-id-pubk-prik* :: $(state \times state)\ set$ **where**
rel-id-pubk-prik $\equiv \{(s,\ s') \mid s\ s'\ n\ S.$
$s' = insert\ (Spy,\ \langle n,\ PubKey\ S\rangle)\ s \wedge$
$\{Tok\text{-}PriKey\ n,\ PubKey\ S\} \subseteq spied\ s \wedge$
$Tok\text{-}PriK\ n \in S\}$

abbreviation *rel-id-hash* :: $(state \times state)\ set$ **where**
rel-id-hash $\equiv \{(s,\ s') \mid s\ s'\ n\ X.$
$s' = s \cup \{Spy\} \times \{\langle n,\ X\rangle,\ \langle n,\ Hash\ X\rangle\} \wedge$
$\{X,\ Hash\ X\} \subseteq spied\ s \wedge$
$(\langle n,\ X\rangle \in spied\ s \vee \langle n,\ Hash\ X\rangle \in spied\ s)\}$

definition *rel* :: $(state \times state)\ set$ **where**
rel $\equiv$ *rel-sector* $\cup$ *rel-token* $\cup$ *rel-pubk-less* $\cup$ *rel-pubk-more* $\cup$
rel-hash $\cup$ *rel-dec* $\cup$ *rel-enc* $\cup$ *rel-sep* $\cup$ *rel-con* $\cup$
rel-id-pubk-less $\cup$ *rel-id-pubk-more* $\cup$ *rel-id-pubk-prik* $\cup$ *rel-id-hash*

abbreviation *in-rel* :: $state \Rightarrow state \Rightarrow bool$ (**infix** $\vdash$ *60*) **where**
$s \vdash s' \equiv (s,\ s') \in rel$

abbreviation *in-rel-rtrancl* :: $state \Rightarrow state \Rightarrow bool$ (**infix** $\models$ *60*) **where**
$s \models s' \equiv (s,\ s') \in rel^*$

end

2 Anonymity of token pseudonymous identifiers

theory *Anonymity*
imports *Definitions*
begin

This section contains a proof of anonymity property *id-anonymous*, which states that a token pseudonymous identifier remains anonymous if its anonymity is not compromised by means other than attacking the protocol and neither attack option described in section 1.2 is viable. As shown here below, this property can be proven by applying rules *rtrancl-induct* and *rtrancl-start* in a suitable combination [7].

proposition *rtrancl-start* [*rule-format*]:
$(x,\ y) \in r^* \Longrightarrow P\ y \longrightarrow \neg\ P\ x \longrightarrow$
$(\exists\, u\ v.\ (x,\ u) \in r^* \wedge (u,\ v) \in r \wedge (v,\ y) \in r^* \wedge \neg\ P\ u \wedge P\ v)$
(**is** - $\Longrightarrow$ - $\longrightarrow$ - $\longrightarrow$ $(\exists\, u\ v.\ ?P_2\ x\ y\ u\ v)$)
⟨*proof*⟩

proposition *state-subset*:
$s \models s' \Longrightarrow s \subseteq s'$
⟨*proof*⟩

proposition *spied-subset*:
$s \models s' \Longrightarrow spied\ s \subseteq spied\ s'$
⟨*proof*⟩

proposition *parts-init*:
$parts\ (used\ s_0) = used\ s_0$
⟨*proof*⟩

proposition *parts-idem* [*simp*]:
$parts\ (parts\ H) = parts\ H$
⟨*proof*⟩

proposition *parts-mono*:
$H \subseteq H' \Longrightarrow parts\ H \subseteq parts\ H'$
⟨*proof*⟩

lemma *parts-union-1*:
$parts\ (H \cup H') \subseteq parts\ H \cup parts\ H'$
⟨*proof*⟩

lemma *parts-union-2*:
$parts\ H \cup parts\ H' \subseteq parts\ (H \cup H')$
⟨*proof*⟩

proposition *parts-union* [*simp*]:
$parts\ (H \cup H') = parts\ H \cup parts\ H'$
⟨*proof*⟩

proposition *parts-insert*:
$parts\ (insert\ X\ H) = parts\text{-}msg\ X \cup parts\ H$
⟨*proof*⟩

proposition *parts-msg-mono*:
$X \in H \Longrightarrow parts\text{-}msg\ X \subseteq parts\ H$
⟨*proof*⟩

proposition *parts-msg-agrkey* [*simp*]:

parts-msg (*AgrKey K*) = {*AgrKey K*}
⟨*proof*⟩

proposition *parts-msg-hash* [*simp*]:
parts-msg (*Hash X*) = {*Hash X*}
⟨*proof*⟩

lemma *parts-crypt-1*:
parts {*Crypt K X*} ⊆ *insert* (*Crypt K X*) (*parts* {*X*})
⟨*proof*⟩

lemma *parts-crypt-2*:
insert (*Crypt K X*) (*parts* {*X*}) ⊆ *parts* {*Crypt K X*}
⟨*proof*⟩

proposition *parts-msg-crypt* [*simp*]:
parts-msg (*Crypt K X*) = *insert* (*Crypt K X*) (*parts-msg X*)
⟨*proof*⟩

lemma *parts-mpair-1*:
parts {⦃*X*, *Y*⦄} ⊆ *insert* ⦃*X*, *Y*⦄ (*parts* {*X*} ∪ *parts* {*Y*})
⟨*proof*⟩

lemma *parts-mpair-2*:
insert ⦃*X*, *Y*⦄ (*parts* {*X*} ∪ *parts* {*Y*}) ⊆ *parts* {⦃*X*, *Y*⦄}
⟨*proof*⟩

proposition *parts-msg-mpair* [*simp*]:
parts-msg ⦃*X*, *Y*⦄ = *insert* ⦃*X*, *Y*⦄ (*parts-msg X* ∪ *parts-msg Y*)
⟨*proof*⟩

proposition *parts-msg-idinfo* [*simp*]:
parts-msg ⟨*n*, *X*⟩ = {⟨*n*, *X*⟩}
⟨*proof*⟩

proposition *parts-msg-parts*:
⟦(*A*, *X*) ∈ *s*; *Y* ∈ *parts-msg X*⟧ ⟹ *Y* ∈ *parts* (*used s*)
⟨*proof*⟩

proposition *prikey-spied*:
⟦*s*$_0$ ⊨ *s*; *PriKey K* ∈ *parts* (*used s*)⟧ ⟹ *PriKey K* ∈ *spied s*
⟨*proof*⟩

proposition *prikey-crypt* [*simplified*]:
⟦(*Spy*, *Crypt K* (*PriKey K′*)) ∈ *s*; *s*$_0$ ⊨ *s*⟧ ⟹ *PriKey K′* ∈ *spied s*
⟨*proof*⟩

proposition *prikey-mpair-fst* [*simplified*]:

$[\![(Spy,\ \{\!|PriKey\ K,\ Y|\!\}) \in s;\ s_0 \models s]\!] \Longrightarrow PriKey\ K \in spied\ s$
⟨*proof*⟩

proposition *prikey-mpair-snd* [*simplified*]:
$[\![(Spy,\ \{\!|Y,\ PriKey\ K|\!\}) \in s;\ s_0 \models s]\!] \Longrightarrow PriKey\ K \in spied\ s$
⟨*proof*⟩

proposition *rev-prikey-secret*:
$s_0 \models s \Longrightarrow Rev\text{-}PriKey \notin spied\ s$
⟨*proof*⟩

proposition *sec-prikey-secret*:
$[\![s_0 \models s;\ n \notin bad\text{-}sec\text{-}prik]\!] \Longrightarrow Sec\text{-}PriKey\ n \notin spied\ s$
⟨*proof*⟩

proposition *tok-prikey-secret*:
$[\![s_0 \models s;\ n \notin bad\text{-}tok\text{-}prik]\!] \Longrightarrow Tok\text{-}PriKey\ n \notin spied\ s$
⟨*proof*⟩

proposition *idinfo-spied*:
$[\![s_0 \models s;\ \langle n,\ X\rangle \in parts\ (used\ s)]\!] \Longrightarrow \langle n,\ X\rangle \in spied\ s$
⟨*proof*⟩

proposition *idinfo-crypt*:
$[\![(Spy,\ Crypt\ K\ \langle n,\ X\rangle) \in s;\ s_0 \models s]\!] \Longrightarrow \langle n,\ X\rangle \in spied\ s$
⟨*proof*⟩

proposition *idinfo-mpair-fst*:
$[\![(Spy,\ \{\!|\langle n,\ X\rangle,\ Y|\!\}) \in s;\ s_0 \models s]\!] \Longrightarrow \langle n,\ X\rangle \in spied\ s$
⟨*proof*⟩

proposition *idinfo-mpair-snd*:
$[\![(Spy,\ \{\!|Y,\ \langle n,\ X\rangle|\!\}) \in s;\ s_0 \models s]\!] \Longrightarrow \langle n,\ X\rangle \in spied\ s$
⟨*proof*⟩

proposition *idinfo-hash-hash* [*rotated*]:
$[\![s_0 \models s;\ (Spy,\ \langle n,\ Hash\ (Hash\ X)\rangle) \in s]\!] \Longrightarrow \langle n,\ Hash\ X\rangle \in spied\ s$
⟨*proof*⟩

proposition *sec-prik-eq*:
$\{Tok\text{-}PriK\ n,\ Sec\text{-}PriK\ m,\ Rev\text{-}PriK\} =$
$\quad \{Tok\text{-}PriK\ n,\ Sec\text{-}PriK\ m',\ Rev\text{-}PriK\} \Longrightarrow m' = m$
⟨*proof*⟩

proposition *id-identified*:
assumes
A: $s_0 \models s$ **and**

B: $(n,\ m) \notin$ *bad-id* **and**
C: $n \notin$ *bad-tok-prik* **and**
D: $\langle n,\ Hash\ (ID\ n\ (Sec\text{-}PubKey\ m))\rangle \in spied\ s$
shows $m \in$ *bad-sec-prik* $\wedge$
$(\exists\, m'.\ m' \neq m \wedge m' \in$ *bad-sec-prik* $\wedge\ (n,\ m') \in$ *bad-id*$)$
⟨*proof*⟩

theorem *id-anonymous* [*rotated*]:
$[\![m \notin$ *bad-sec-prik* $\vee \neg\ (\exists\, m'.\ m' \neq m \wedge m' \in$ *bad-sec-prik* $\wedge\ (n,\ m') \in$ *bad-id*$)$;
$s_0 \models s$; $(n,\ m) \notin$ *bad-id*; $n \notin$ *bad-tok-prik*$]\!] \Longrightarrow$
$\langle n,\ Hash\ (ID\ n\ (Sec\text{-}PubKey\ m))\rangle \notin spied\ s$
⟨*proof*⟩

end

3 Possibility of anonymity compromise for token pseudonymous identifiers

theory *Possibility*
imports *Anonymity*
begin

This section proves possibility properties *tok-id-identified*, *sec-id-identified*, which altogether state that the spy can map a token pseudonymous identifier to the related token if either attack option described in section 1.2 is viable. Both properties are proven by construction, namely by creating as many sample protocol runs such as to satisfy their conclusions if their assumptions are fulfilled.

definition *tok-id-pubk-prik* :: *agent-id* ⇒ *agent-id* ⇒ *state* **where**
tok-id-pubk-prik n m ≡
insert $(Spy,\ \langle n,\ ID\ n\ (Sec\text{-}PubKey\ m)\rangle)\ s_0$

definition *tok-id-hash* :: *agent-id* ⇒ *agent-id* ⇒ *state* **where**
tok-id-hash n m ≡
insert $(Spy,\ \langle n,\ Hash\ (ID\ n\ (Sec\text{-}PubKey\ m))\rangle)$ *(tok-id-pubk-prik n m)*

proposition *tok-id-pubk-prik-rel*:
$n \in$ *bad-tok-prik* $\Longrightarrow s_0 \models$ *tok-id-pubk-prik n m*
⟨*proof*⟩

proposition *tok-id-pubk-prik-msg*:
$n \in$ *bad-tok-prik* $\Longrightarrow$
$\{ID\ n\ (Sec\text{-}PubKey\ m),\ Hash\ (ID\ n\ (Sec\text{-}PubKey\ m)),$

$\langle n,\ ID\ n\ (Sec\text{-}PubKey\ m)\rangle\} \subseteq spied\ (tok\text{-}id\text{-}pubk\text{-}prik\ n\ m)$
⟨*proof*⟩

proposition *tok-id-hash-rel*:
$n \in bad\text{-}tok\text{-}prik \Longrightarrow s_0 \models tok\text{-}id\text{-}hash\ n\ m$
⟨*proof*⟩

theorem *tok-id-identified*:
$n \in bad\text{-}tok\text{-}prik \Longrightarrow \exists\, s.\ s_0 \models s \wedge \langle n,\ Hash\ (ID\ n\ (Sec\text{-}PubKey\ m))\rangle \in spied\ s$
⟨*proof*⟩

definition *sec-pubk-less* :: *agent-id* ⇒ *state* **where**
sec-pubk-less n ≡
  insert (*Spy*, *PubKey* {*Tok-PriK n*, *Rev-PriK*}) s_0

definition *sec-id-pubk-less* :: *agent-id* ⇒ *state* **where**
sec-id-pubk-less n ≡
  insert (*Spy*, ⟨*n*, *PubKey* {*Tok-PriK n*, *Rev-PriK*}⟩) (*sec-pubk-less n*)

definition *sec-id-pubk-more* :: *agent-id* ⇒ *agent-id* ⇒ *state* **where**
sec-id-pubk-more n m ≡
  insert (*Spy*, ⟨*n*, *ID n* (*Sec-PubKey m*)⟩) (*sec-id-pubk-less n*)

definition *sec-id-hash* :: *agent-id* ⇒ *agent-id* ⇒ *state* **where**
sec-id-hash n m ≡
  insert (*Spy*, ⟨*n*, *Hash* (*ID n* (*Sec-PubKey m*))⟩) (*sec-id-pubk-more n m*)

lemma *sec-id-identified-1*:
{*Tok-PriK n*, *Sec-PriK m*, *Rev-PriK*} ≠ {*Tok-PriK n′*, *Rev-PriK*}
⟨*proof*⟩

lemma *sec-id-identified-2*:
(*Spy*, *PubKey* {*Tok-PriK n*, *Rev-PriK*}) ∉ s_0
⟨*proof*⟩

lemma *sec-id-identified-3*:
{*Tok-PriK n*, *Rev-PriK*} =
  {*Tok-PriK n*, *Sec-PriK m*, *Rev-PriK*} − {*Sec-PriK m*}
⟨*proof*⟩

lemma *sec-id-identified-4*:
PubK {*Tok-PriK n*, *Sec-PriK m*, *Rev-PriK*} =
  PubK (*insert* (*Sec-PriK m*) {*Tok-PriK n*, *Rev-PriK*})
⟨*proof*⟩

proposition *sec-pubk-less-rel*:

$[\![\{m,\ m'\} \subseteq \mathit{bad\text{-}sec\text{-}prik};\ (n,\ m) \notin \mathit{bad\text{-}id};\ (n,\ m') \in \mathit{bad\text{-}id}]\!] \Longrightarrow$
$\quad s_0 \models \mathit{sec\text{-}pubk\text{-}less}\ n$
⟨*proof*⟩

proposition *sec-pubk-less-msg*:
$[\![\{m,\ m'\} \subseteq \mathit{bad\text{-}sec\text{-}prik};\ (n,\ m) \notin \mathit{bad\text{-}id};\ (n,\ m') \in \mathit{bad\text{-}id}]\!] \Longrightarrow$
$\quad \{\mathit{Sec\text{-}PriKey}\ m,\ \mathit{Sec\text{-}PriKey}\ m',\ \mathit{PubKey}\ \{\mathit{Tok\text{-}PriK}\ n,\ \mathit{Rev\text{-}PriK}\},$
$\quad\ \mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m),\ \mathit{Hash}\ (\mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m)),$
$\quad\ \langle n,\ \mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m')\rangle\} \subseteq \mathit{spied}\ (\mathit{sec\text{-}pubk\text{-}less}\ n) \wedge$
$\quad \{\langle n,\ \mathit{PubKey}\ \{\mathit{Tok\text{-}PriK}\ n,\ \mathit{Rev\text{-}PriK}\}\rangle,\ \langle n,\ \mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m)\rangle,$
$\quad\ \langle n,\ \mathit{Hash}\ (\mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m))\rangle\} \cap$
$\quad\ \mathit{spied}\ (\mathit{sec\text{-}pubk\text{-}less}\ n) = \{\}$
⟨*proof*⟩

proposition *sec-id-pubk-less-rel*:
$[\![\{m,\ m'\} \subseteq \mathit{bad\text{-}sec\text{-}prik};\ (n,\ m) \notin \mathit{bad\text{-}id};\ (n,\ m') \in \mathit{bad\text{-}id}]\!] \Longrightarrow$
$\quad s_0 \models \mathit{sec\text{-}id\text{-}pubk\text{-}less}\ n$
⟨*proof*⟩

proposition *sec-id-pubk-less-msg*:
$[\![\{m,\ m'\} \subseteq \mathit{bad\text{-}sec\text{-}prik};\ (n,\ m) \notin \mathit{bad\text{-}id};\ (n,\ m') \in \mathit{bad\text{-}id}]\!] \Longrightarrow$
$\quad \{\mathit{Sec\text{-}PriKey}\ m,\ \mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m),\ \mathit{Hash}\ (\mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m)),$
$\quad\ \langle n,\ \mathit{PubKey}\ \{\mathit{Tok\text{-}PriK}\ n,\ \mathit{Rev\text{-}PriK}\}\rangle\} \subseteq$
$\quad\ \mathit{spied}\ (\mathit{sec\text{-}id\text{-}pubk\text{-}less}\ n) \wedge$
$\quad \{\langle n,\ \mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m)\rangle,\ \langle n,\ \mathit{Hash}\ (\mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m))\rangle\} \cap$
$\quad\ \mathit{spied}\ (\mathit{sec\text{-}id\text{-}pubk\text{-}less}\ n) = \{\}$
⟨*proof*⟩

proposition *sec-id-pubk-more-rel*:
$[\![\{m,\ m'\} \subseteq \mathit{bad\text{-}sec\text{-}prik};\ (n,\ m) \notin \mathit{bad\text{-}id};\ (n,\ m') \in \mathit{bad\text{-}id}]\!] \Longrightarrow$
$\quad s_0 \models \mathit{sec\text{-}id\text{-}pubk\text{-}more}\ n\ m$
⟨*proof*⟩

proposition *sec-id-pubk-more-msg*:
$[\![\{m,\ m'\} \subseteq \mathit{bad\text{-}sec\text{-}prik};\ (n,\ m) \notin \mathit{bad\text{-}id};\ (n,\ m') \in \mathit{bad\text{-}id}]\!] \Longrightarrow$
$\quad \{\mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m),\ \mathit{Hash}\ (\mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m)),$
$\quad\ \langle n,\ \mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m)\rangle\} \subseteq \mathit{spied}\ (\mathit{sec\text{-}id\text{-}pubk\text{-}more}\ n\ m) \wedge$
$\quad \langle n,\ \mathit{Hash}\ (\mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m))\rangle \notin \mathit{spied}\ (\mathit{sec\text{-}id\text{-}pubk\text{-}more}\ n\ m)$
⟨*proof*⟩

proposition *sec-id-hash-rel*:
$[\![\{m,\ m'\} \subseteq \mathit{bad\text{-}sec\text{-}prik};\ (n,\ m) \notin \mathit{bad\text{-}id};\ (n,\ m') \in \mathit{bad\text{-}id}]\!] \Longrightarrow$
$\quad s_0 \models \mathit{sec\text{-}id\text{-}hash}\ n\ m$
⟨*proof*⟩

theorem *sec-id-identified*:
$[\![\{m,\ m'\} \subseteq \mathit{bad\text{-}sec\text{-}prik};\ (n,\ m') \in \mathit{bad\text{-}id}]\!] \Longrightarrow$
$\quad \exists\, s.\ s_0 \models s \wedge \langle n,\ \mathit{Hash}\ (\mathit{ID}\ n\ (\mathit{Sec\text{-}PubKey}\ m))\rangle \in \mathit{spied}\ s$

⟨*proof*⟩

end

References

[1] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Technical Guideline TR-03110 – Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token – Part 2: Protocols for electronic IDentification, Authentication and trust Services (eIDAS), version 2.21*, Dec. 2016.

[2] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Technical Guideline TR-03110 – Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token – Part 3: Common Specifications, version 2.21*, Dec. 2016.

[3] A. Krauss. *Defining Recursive Functions in Isabelle/HOL*. https://isabelle.in.tum.de/website-Isabelle2021/dist/Isabelle2021/doc/functions.pdf.

[4] T. Nipkow. *A Tutorial Introduction to Structured Isar Proofs*. https://isabelle.in.tum.de/website-Isabelle2011/dist/Isabelle2011/doc/isar-overview.pdf.

[5] T. Nipkow. *Programming and Proving in Isabelle/HOL*, Feb. 2021. https://isabelle.in.tum.de/website-Isabelle2021/dist/Isabelle2021/doc/prog-prove.pdf.

[6] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, Feb. 2021. https://isabelle.in.tum.de/website-Isabelle2021/dist/Isabelle2021/doc/tutorial.pdf.

[7] P. Noce. The Relational Method with Message Anonymity for the Verification of Cryptographic Protocols. *Archive of Formal Proofs*, Dec. 2020. https://isa-afp.org/entries/Relational_Method.html, Formal proof development.