The Relational Method with Message Anonymity for the Verification of Cryptographic Protocols

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

March 17, 2025

Abstract

This paper introduces a new method for the formal verification of cryptographic protocols, the relational method, derived from Paulson's inductive method by means of some enhancements aimed at streamlining formal definitions and proofs, specially for protocols using public key cryptography. Moreover, this paper proposes a method to formalize a further security property, message anonymity, in addition to message confidentiality and authenticity.

The relational method, including message anonymity, is then applied to the verification of a sample authentication protocol, comprising Password Authenticated Connection Establishment (PACE) with Chip Authentication Mapping followed by the explicit verification of an additional password over the PACE secure channel.

Contents

1   The relational method and message anonymity

theory *Definitions*

imports *Main*
begin

This paper is dedicated to my mother, my favourite chess opponent – in addition to being many other wonderful things!

1.1 Introduction

As Bertrand Russell says in the last pages of *A History of Western Philosophy*, a distinctive feature of science is that "we can make successive approximations to the truth, in which each new stage results from an improvement, not a rejection, of what has gone before". When dealing with a formal verification method for information processing systems, such as Paulson's inductive method for the verification of cryptographic protocols (cf. [7], [5]), a more modest goal for this iterative improvement process, yet of significant practical importance, is to streamline the definitions and proofs needed to model such a system and verify its properties.

With this aim, specially when it comes to verifying protocols using public key cryptography, this paper proposes an enhancement of the inductive method, named *relational method* for reasons clarified in what follows, and puts it into practice by verifying a sample protocol. This new method is the result of some changes to the way how events, states, spy's capabilities, and the protocol itself are formalized in the inductive method. Here below is a description of these changes, along with a rationale for them.

Events. In the inductive method, the fundamental building blocks of cryptographic protocols are events of the form *Says A B X*, where *X* is a message being exchanged, *A* is the agent that sends it, and *B* is the agent to which it is addressed.

However, any exchanged message can be intercepted by the spy and forwarded to any other agent, so its intended recipient is not relevant for the protocol *security* correctness – though of course being relevant for the protocol *functional* correctness. Moreover, a legitimate agent may also generate messages, e.g. ephemeral private keys, that she will never exchange with any other agent. To model such an event, a datatype constructor other than *Says* should be used. How to make things simpler?

The solution adopted in the relational method is to model events just as ordered pairs of the form (*A*, *X*), where *A* is an agent and *X* is a message. If event (*A*, *X*) stands for *A*'s sending of *X* to another agent, where *A* is a legitimate agent, then this event will be accompanied by event (*Spy*, *X*), representing the spy's interception of *X*. If event (*A*, *X*) rather stands for *A*'s generation of private message *X*,

e.g. an ephemeral private key, for her own exclusive use – and if the spy has not hacked A so as to steal her private messages as well –, then no companion event (Spy, X) will occur instead.

States. In the inductive method, the possible states of a cryptographic protocol are modeled as event *traces*, i.e. lists, and the protocol itself is formalized as a set of such traces. Consequently, the protocol rules and security properties are expressed as formulae satisfied by any event trace *evs* belonging to this set.

However, these formulae are such that their truth values depend only on the events contained in *evs*, rather than on the actual order in which they occur – in fact, robust protocol rules and security properties cannot depend on the exact sequence of message exchanges in a scenario where the spy can freely intercept and forward messages, or even generate and send her own ones. Thus, one library function, *set*, and two custom recursive functions, *used* and *knows*, are needed to convert event traces into event sets and message sets, respectively. In the relational method, protocol states are simply modeled as event sets, so that the occurrence of event (A, X) in state s can be expressed as the transition to the augmented state *insert* (A, X) s. Hence, states consist of relations between agents and messages. As a result, function *set* need not be used any longer, whereas functions *used* and *spied* – the latter one being a replacement for *knows Spy* –, which take a state s as input, are mere abbreviations for *Range s* and *s `` {Spy}*.

Spy's capabilities. In the inductive method, the spy's attack capabilities are formalized via two inductively defined functions, *analz* and *synth*, used to construct the sets of all the messages that the spy can learn – *analz* (*knows Spy evs*) – and send to legitimate agents – *synth* (*analz* (*knows Spy evs*)) – downstream of event trace *evs*.

Indeed, the introduction of these functions goes in the direction of decoupling the formalization of the spy's capabilities from that of the protocol itself, consistently with the fact that what the spy can do is independent of how the protocol works – which only matters when it comes to verifying protocol security.

In principle, this promises to provide a relevant benefit: these functions need to be defined, and their properties to be proven, just once, whereupon such definitions and properties can be reused in the formalization and verification of whatever protocol.

In practice, since both functions are of type *msg set* $\Rightarrow$ *msg set*, where *msg* is the datatype defining all possible message formats, this benefit only applies as long as message formats remain unchanged. However, when it comes to verifying a protocol making use of public key cryptography, some new message format, and consequently some new related

3

spy's capability as well, are likely to be required. An example of this will be provided right away by the protocol considered in this paper.

In the relational method, the representation of events as agent-message pairs offers a simpler way to model the spy's capabilities, namely as supplementary protocol rules, analogous to the inductive method's *Fake* rule, augmenting a state by one or more events of the form (Spy, X). In addition to eliminating the need for functions *analz* and *synth* – which, in light of the above considerations, does not significantly harm reusability –, this choice also abolishes any distinction between what the spy can learn and what she can send. In fact, a state containing event (Spy, X) is interpreted as one where the spy both knows message X and may have sent it to whatever legitimate agent. Actually, this formalizes the facts that a real-world attacker is free to send any message she has learned to any other party, and conversely to use any message she has generated to further augment her knowledge.

In the inductive method, the former fact is modeled by property $H \subseteq synth\ H$ of function *synth*, but the latter one has no formal counterpart, as in general $H \subset synth\ H$. This limitation on the spy's capabilities is not significant as long as the protocol makes use of static keys only, but it is if session keys or ephemeral key pairs are generated – as happens in key establishment protocols, even in those using symmetric cryptography alone. In any such case, a realistic spy must also be able to learn from anything she herself has generated, such as a nonce or an ephemeral private key – a result achieved without effort in the relational method.

An additional, nontrivial problem for the inductive method is that many protocols, including key establishment ones, require the spy to be able to generate *fresh* ephemeral messages only, as otherwise the spy could succeed in breaking the protocol by just guessing the ephemeral messages already generated at random by some legitimate agent – a quite unrealistic attack pattern, provided that such messages vary in a sufficiently wide range. At first glance, this need could be addressed by extending the inductive definition of function *synth* with introduction rules of the form *Nonce* $n \notin H \implies$ *Nonce* $n \in synth\ H$ or *PriKey* $A \notin H \implies$ *PriKey* $A \in synth\ H$. However, private ephemeral messages are not in general included in *analz* (*knows Spy evs*), since nonces may be encrypted with uncompromised keys when exchanged and private keys are usually not exchanged at all, so this approach would not work. The only satisfactory alternative would be to change the signature of function *synth*, e.g. by adding a second input message set H' standing for *used evs*, or else by replacing H with event trace *evs* itself, but this would render the function definition much more convoluted – a problem easily bypassed in the relational method.

Protocol. In the inductive method, a cryptographic protocol consists of an inductively defined set of event traces. This enables to prove the protocol security properties by induction using the induction rule automatically generated as a result of such an inductive definition, i.e. by means of *rule induction*. Actually, this feature is exactly what gives the method its very name. Hence, a consistent way to name a protocol verification method using some other form of induction would be to replace adjective "inductive" with another one referring to that form of induction.

The relational method owes its name to this consideration. In this method, the introduction rules defining *protocol rules*, i.e. the possible transitions between protocol states, are replaced with *relations* between states, henceforth named *protocol relations*. That is, for any two states s and s', there exists a transition leading from s to s' just in case the ordered pair (s, s') is contained in at least one protocol relation – a state of affairs denoted using infix notation $s \vdash s'$. Then, the inductively defined set itself is replaced with the *reflexive transitive closure* of the union of protocol relations. Namely, any state s may be reached from *initial state* s_0, viz. is a possible protocol state, just in case pair (s_0, s) lies within this reflexive transitive closure – a state of affairs denoted using infix notation $s_0 \models s$. As a result, rule induction is replaced with induction over reflexive transitive closures via rule *rtrancl-induct*, which is the circumstance that originates the method name.

These changes provide the following important benefits.

- Inserting and modifying the formal definition of a protocol is much more comfortable. In fact, any change even to a single introduction rule within a monolithic inductive set definition entails a re-evaluation of the whole definition, whereas each protocol relation will have its own stand-alone definition, which also makes it easier to find errors. This advantage may go almost unnoticed for a very simple protocol providing for just a few protocol rules, but gets evident in case of a complex protocol. An example of this will be provided by the protocol considered in this paper: when looking at the self-contained abbreviations used to define protocol relations, the reader will easily grasp how much more convoluted an equivalent inductive set definition would have been.

- In addition to induction via rule *rtrancl-induct*, a further powerful reasoning pattern turns out to be available. It is based on the following general rule applying to reflexive transitive closures (indeed, a rule so general and useful that it could rightfully become part of the standard library), later on proven and assigned the name *rtrancl-start*:

$$\llbracket(x,\ y)\in r^*;\ P\ y;\ \neg\ P\ x\rrbracket$$
$$\Longrightarrow \exists\,u\ v.\ (x,\ u)\in r^* \wedge (u,\ v)\in r \wedge (v,\ y)\in r^* \wedge \neg\ P\ u \wedge P\ v$$

In natural language, this rule states that for any chain of elements linked by a relation, if some predicate is false for the first element of the chain and true for the last one, there must exist a link in the chain where the predicate becomes true.

This rule can be used to prove propositions of the form $\llbracket s \models s'$; $P\ s'\rrbracket;\ Q\rrbracket \Longrightarrow R\ s'$ for any state s and predicate P such that $\neg\ P\ s$, with an optional additional assumption Q, without resorting to induction. Notably, *regularity lemmas* have exactly this form, where $s = s_0$, $P = (\lambda s.\ X \in parts\ (used\ s))$ for some term X of type *msg*, and Q, if present, puts some constraint on X or its components.

Such a proof consists of two steps. First, lemma $\llbracket s \vdash s'$; $P\ s'$; $\neg\ P\ s\rrbracket;\ Q\rrbracket \Longrightarrow R\ s'$ is proven by simplification, using the definitions of protocol relations. Then, the target proposition is proven by applying rule *rtrancl-start* as a destruction rule (cf. [5]) and proving $P\ s'$ by assumption, $\neg\ P\ s$ by simplification, and the residual subgoal by means of the previous lemma.

In addition to the relational method, this paper is aimed at introducing still another enhancement: besides message confidentiality and authenticity, it takes into consideration a further important security property, *message anonymity*. Being legitimate agents identified via natural numbers, the fact that in state s the spy ignores that message X_n is associated with agent n, viz. X_n's property of being *anonymous* in state s, can be expressed as $\langle n,\ X_n \rangle \notin spied\ s$, where notation $\langle n,\ X_n \rangle$ refers to a new constructor added to datatype *msg* precisely for this purpose.

A basic constraint upon any protocol relation augmenting the spy's knowledge with $\langle n,\ X \rangle$ is that the spy must know message X in the current state, as it is impossible to identify the agent associated with an unknown message. There is also an additional, more subtle constraint. Any such protocol relation either augments a state in which the spy knows $\langle n,\ C\ X_1\ \ldots\ X_m \rangle$, i.e. containing event $(Spy,\ \langle n,\ C\ X_1\ \ldots\ X_m \rangle)$, with event $(Spy,\ \langle n,\ X_i \rangle)$, where $1 \leq i \leq m$ and C is some constructor of datatype *msg*, or conversely augments a state containing event $(Spy,\ \langle n,\ X_i \rangle)$ with $(Spy,\ \langle n,\ C\ X_1\ \ldots\ X_m \rangle)$. However, the latter spy's inference is justified only if the compound message $C\ X_1\ \ldots\ X_m$ is part of a message generated or accepted by some legitimate agent according to the protocol rules. Otherwise, that is, if $C\ X_1\ \ldots\ X_m$ were just a message generated at random by the spy, her inference would be as sound as those of most politicians and all advertisements: even if the conclusion were true, it would be so by pure chance.

This problem can be solved as follows.

- A further constructor *Log*, taking a message as input, is added to datatype *msg*, and every protocol relation modeling the generation or acceptance of a message *X* by some legitimate agent must augment the current state with event (*Spy*, *Log X*).

  In this way, the set of all the messages that have been generated or accepted by some legitimate agent in state *s* matches *Log −' spied s*.

- A function *crypts* is defined inductively. It takes a message set *H* as input, and returns the least message set *H′* such that *H* ⊆ *H′* and for any (even empty) list of keys *KS*, if the encryption of $\{\!|X,\ Y|\!\}$, $\{\!|Y,\ X|\!\}$, or *Hash X* with *KS* is contained in *H′*, then the encryption of *X* with *KS* is contained in *H′* as well.

  In this way, the set of all the messages that are part of messages exchanged by legitimate agents, viz. that may be mapped to agents, in state *s* matches *crypts* (*Log −' spied s*).

- Another function *key-sets* is defined, too. It takes two inputs, a message *X* and a message set *H*, and returns the set of the sets of *KS*' inverse keys for any list of keys *KS* such that the encryption of *X* with *KS* is included in *H*.

  In this way, the fact that in state *s* the spy can map a compound message *X* to some agent, provided that she knows all the keys in set *U*, can be expressed through conditions *U* ∈ *key-sets X* (*crypts* (*Log −' spied s*)) and *U* ⊆ *spied s*.

  The choice to define *key-sets* so as to collect the inverse keys of encryption keys, viz. decryption ones, depends on the fact that the sample protocol verified in this paper uses symmetric keys alone – which match their own inverse keys – for encryption, whereas asymmetric key pairs are used in cryptograms only for signature generation – so that the inverse keys are public ones. In case of a protocol (also) using public keys for encryption, encryption keys themselves should (also) be collected, since the corresponding decryption keys, i.e. private keys, would be unknown to the spy by default. This would formalize the fact that encrypted messages can be mapped to agents not only by decrypting them, but also by recomputing the cryptograms (provided that the plaintexts are known) and checking whether they match the exchanged ones.

1.2 A sample protocol

As previously mentioned, this paper tries the relational method, including message anonymity, by applying it to the verification of a sample authentication protocol in which Password Authenticated Connection Establishment (PACE) with Chip Authentication Mapping (cf. [1]) is first used by an *owner*

to establish a secure channel with her own *asset* and authenticate it, and then the owner sends a password (other than the PACE one) to the asset over that channel so as to authenticate herself. This enables to achieve a reliable mutual authentication even if the PACE key is shared by multiple owners or is weak, as happens in electronic passports. Although the PACE mechanism is specified for use in electronic documents, nothing prevents it in principle from being used in other kinds of smart cards or even outside of the smart card world, which is the reason why this paper uses the generic names *asset* and *owner* for the card and the cardholder, respectively.

In more detail, this protocol provides for the following steps. In this list, messages are specified using the same syntax that will be adopted in the formal text (for further information about PACE with Chip Authentication Mapping, cf. [1]).

1. *Asset n → Owner n*:
   Crypt (*Auth-ShaKey n*) (*PriKey S*)

2. *Owner n → Asset n*:
   {|*Num 1*, *PubKey A*|}

3. *Asset n → Owner n*:
   {|*Num 2*, *PubKey B*|}

4. *Owner n → Asset n*:
   {|*Num 3*, *PubKey C*|}

5. *Asset n → Owner n*:
   {|*Num 4*, *PubKey D*|}

6. *Owner n → Asset n*:
   Crypt (*SesK SK*) (*PubKey D*)

7. *Asset n → Owner n*:
   {|*Crypt* (*SesK SK*) (*PubKey C*),
     Crypt (*SesK SK*) (*Auth-PriK n* ⊗ *B*),
     Crypt (*SesK SK*) (*Crypt SigK*
       {|*Hash* (*Agent n*), *Hash* (*Auth-PubKey n*)|})|}

8. *Owner n → Asset n*:
   Crypt (*SesK SK*) (*Pwd n*)

9. *Asset n → Owner n*:
   Crypt (*SesK SK*) (*Num 0*)

Legitimate agents consist of an infinite population of assets and owners. For each natural number *n*, *Owner n* is an owner and *Asset n* is her own asset, and these agents are assigned the following authentication data.

- *Key* (*Auth-ShaKey n*): static symmetric PACE key shared by both agents.

- *Auth-PriKey n*, *Auth-PubKey n*: static private and public keys stored on *Asset n* and used for *Asset n*'s authentication via Chip Authentication Mapping.

- *Pwd n*: unique password (other than the PACE one) shared by both agents and used for *Owner n*'s authentication.

Function *Pwd* is defined as a constructor of datatype *msg* and then is injective, which formalizes the assumption that each asset-owner pair has a distinct password, whereas no such constraint is put on functions *Auth-ShaKey*, *Auth-PriKey*, and *Auth-PubKey*, which allows multiple asset-owner pairs to be assigned the same keys. On the other hand, function *Auth-PriKey* is constrained to be such that the complement of its range is infinite. As each protocol run requires the generation of fresh ephemeral private keys, this constraint ensures that an unbounded number of protocol runs can be carried out. All assumptions are formalized by applying the definitional approach, viz. without introducing any axiom, and so is this constraint, expressed by defining function *Auth-PriKey* using the indefinite description operator *SOME*.

The protocol starts with *Asset n* sending an ephemeral private key encrypted with the PACE key to *Owner n*. Actually, if *Asset n* is a smart card, the protocol should rather start with *Owner n* sending a plain request for such encrypted nonce, but this preliminary step is omitted here as it is irrelevant for protocol security. After that, *Owner n* and *Asset n* generate two ephemeral key pairs each and send the respective public keys to the other party.

Then, both parties agree on the same session key by deriving it from the ephemeral keys generated previously (actually, two distinct session keys would be derived, one for encryption and the other one for MAC computation, but such a level of detail is unnecessary for protocol verification). The session key is modeled as *Key* (*SesK SK*), where *SesK* is an apposite constructor added to datatype *key* and *SK* = (*Some S*, {*A*, *B*}, {*C*, *D*}). The adoption of type *nat option* for the first component enables to represent as (*None*, {*A*, *B*}, {*C*, *D*}) the wrong session key derived from *Owner n* if *PriKey S* was encrypted using a key other than *Key* (*Auth-ShaKey n*) – which reflects the fact that the protocol goes on even without the two parties sharing the same session key. The use of type *nat set* for the other two components enables the spy to compute *Key* (*SesK SK*) if she knows *either* private key and the other public key referenced by each set, as long as she also knows *PriKey S* – which reflects the fact that given two key pairs, Diffie-Hellman key agreement generates the same shared secret independently of which of the respective private keys is used for computation.

This session key is used by both parties to compute their authentication tokens. Both encrypt the other party's second ephemeral public key, but *Asset n* appends two further fields: the Encrypted Chip Authentication Data, as provided for by Chip Authentication Mapping, and an encrypted signature of the hash values of *Agent n* and *Auth-PubKey n*. Infix notation *Auth-PriK n ⊗ B* refers to a constructor of datatype *msg* standing for plain Chip Authentication Data, and *Agent* is another such constructor standing for agent identification data. *Owner n* is expected to validate this signature by also checking *Agent n*'s hash value against reference identification data known by other means – otherwise, the spy would not be forced to know *Auth-PriKey n* to masquerade as *Asset n*, since she could do that by just knowing *Auth-PriKey m* for some other *m*, even if *Auth-PriKey m* ≠ *Auth-PriKey n*. If *Asset n* is an electronic passport, the owner, i.e. the inspection system, could get cardholder's identification data by reading her personal data on the booklet, and such a signature could be retrieved from the chip (actually through a distinct message, but this is irrelevant for protocol security as long as the password is sent after the signature's validation) by reading the Document Security Object – provided that *Auth-PubKey n* is included within Data Group 14.

The protocol ends with *Owner n* sending her password, encrypted with the session key, to *Asset n*, who validates it and replies with an encrypted acknowledgment.

Here below are some concluding remarks about the way how this sample protocol is formalized.

- A single signature private key, unknown to the spy, is assumed to be used for all legitimate agents. Similarly, the spy might have hacked some legitimate agent so as to steal her ephemeral private keys and session keys as soon as they are generated, but here all legitimate agents are assumed to be out of the spy's reach in this respect. Of course, this is just the choice of one of multiple possible modeling scenarios, and nothing prevents these assumptions from being dropped.

- In the real world, a legitimate agent would use any one of her ephemeral private keys just once, after which the key would be destroyed. On the contrary, no such constraint is enforced here, since it turns out to be unnecessary for protocol verification. There is a single exception, required for the proof of a unicity lemma: after *Asset n* has used *PriKey B* to compute her authentication token, she must discard *PriKey B* so as not to use this key any longer. The way how this requirement is expressed emphasizes once more the flexibility of the modeling of events in the relational method: *Asset n* may use *PriKey B* in this computation only if event (*Asset n*, *PubKey B*) is not yet contained in the current state *s*, and then *s* is augmented with that event. Namely,

events can also be used to model garbage collection!

- The sets of the legitimate agents whose authentication data have been identified in advance (or equivalently, by means other than attacking the protocol, e.g. by social engineering) by the spy are defined consistently with the constraint that known data alone can be mapped to agents, as well as with the definition of initial state s_0. For instance, the set *bad-id-prikey* of the agents whose Chip Authentication private keys have been identified is defined as a subset of the set *bad-prikey* of the agents whose Chip Authentication private keys have been stolen. Moreover, all the signatures included in assets' authentication tokens are assumed to be already known to the spy in state s_0, so that *bad-id-prikey* includes also any agent whose identification data or Chip Authentication public key have been identified in advance.

- The protocol rules augmenting the spy's knowledge with some message of the form $\langle n, X \rangle$ generally require the spy to already know some other message of the same form. There is just one exception: the spy can infer $\langle n, Agent\ n \rangle$ from *Agent n*. This expresses the fact that the detection of identification data within a message generated or accepted by some legitimate agent is in itself sufficient to map any known component of that message to the identified agent, regardless of whether any data were already mapped to that agent in advance.

- As opposed to what happens for constructors ($\otimes$) and *MPair*, there do not exist two protocol rules enabling the spy to infer $\langle n, Crypt\ K\ X \rangle$ from $\langle n, X \rangle$ or $\langle n, Key\ K \rangle$ and vice versa. A single protocol rule is rather defined, which enables the spy to infer $\langle n, X \rangle$ from $\langle n, Key\ K \rangle$ or vice versa, provided that *Crypt K X* has been exchanged by some legitimate agent. In fact, the protocol provides for just one compound message made up of cryptograms, i.e. the asset's authentication token, and all these cryptograms are generated using the same encryption key *Key* (*SesK SK*). Thus, if two such cryptograms have plaintexts X_1, X_2 and the spy knows $\langle n, X_1 \rangle$, she can infer $\langle n, X_2 \rangle$ by inferring $\langle n, Key\ (SesK\ SK) \rangle$, viz. she need not know $\langle n, Crypt\ (SesK\ SK)\ X_1 \rangle$ to do that.

The formal content is split into the following sections.

- Section 1.3, *Definitions*, contains all the definitions needed to formalize the sample protocol by means of the relational method, including message anonymity.

- Section 2, *Confidentiality and authenticity properties*, proves that the following theorems hold under appropriate assumptions.

11

1. Theorem *sigkey-secret*: the signature private key is secret.

2. Theorem *auth-shakey-secret*: an asset-owner pair's PACE key is secret.

3. Theorem *auth-prikey-secret*: an asset's Chip Authentication private key is secret.

4. Theorem *owner-seskey-unique*: an owner's session key is unknown to other owners.

5. Theorem *owner-seskey-secret*: an owner's session key is secret.

6. Theorem *owner-num-genuine*: the encrypted acknowledgment received by an owner has been sent by the respective asset.

7. Theorem *owner-token-genuine*: the PACE authentication token received by an owner has been generated by the respective asset, using her Chip Authentication private key and the same ephemeral keys used to derive the session key.

8. Theorem *pwd-secret*: an asset-owner pair's password is secret.

9. Theorem *asset-seskey-unique*: an asset's session key is unknown to other assets, and may be used by that asset to compute just one PACE authentication token.

10. Theorem *asset-seskey-secret*: an asset's session key is secret.

11. Theorem *asset-pwd-genuine*: the encrypted password received by an asset has been sent by the respective owner.

12. Theorem *asset-token-genuine*: the PACE authentication token received by an asset has been generated by the respective owner, using the same ephemeral key used to derive the session key.

13. Theorem *seskey-forward-secret*: a session key shared by an asset-owner pair is endowed with *forward secrecy*, viz. it is secret independently of the secrecy of static keys.

Particularly, these proofs confirm that the mutual authentication between an owner and her asset is reliable even if their PACE key is compromised, unless either their Chip Authentication private key or their password also is – namely, the protocol succeeds in implementing a two-factor mutual authentication –, with the forward secrecy of the generated session keys being ensured as well.

- Section 3, *Anonymity properties*, proves that the following theorems hold under appropriate assumptions.

  1. Theorem *pwd-anonymous*: an asset-owner pair's password is anonymous.

2. Theorem *auth-prikey-anonymous*: an asset's Chip Authentication private key is anonymous.

3. Theorem *auth-shakey-anonymous*: an asset-owner pair's PACE key is anonymous.

- Section 4, *Possibility properties*, shows how possibility properties (cf. [7]) can be proven by constructing sample protocol runs, either ordinary or attack ones. Two such properties are proven:

  1. Theorem *runs-unbounded*: for any possible protocol state s and any asset-owner pair, there exists a state s' reachable from s in which a protocol run has been completed by those agents using an ephemeral private key *PriKey S* not yet exchanged in s – namely, an unbounded number of protocol runs can be carried out by legitimate agents.

  2. Theorem *pwd-compromised*: in a scenario not satisfying the assumptions of theorem *pwd-anonymous*, the spy can steal an asset-owner pair's password and even identify those agents.

  The latter is an example of a possibility property aimed at confirming that the assumptions of a given confidentiality, authenticity, or anonymity property are necessary for it to hold.

For further information about the formal definitions and proofs contained in these sections, see Isabelle documentation, particularly [5], [4], [2], and [3].

Important note. This sample protocol was already considered in a former paper of mine (cf. [6]). For any purpose, that paper should be regarded as being obsolete and superseded by the present paper.

1.3 Definitions

type-synonym *agent-id = nat*

type-synonym *key-id = nat*

type-synonym *seskey-in = key-id option × key-id set × key-id set*

datatype *agent =*
  Asset agent-id |
  Owner agent-id |
  Spy

datatype *key =*
  SigK |
  VerK |

PriK key-id |
PubK key-id |
ShaK key-id |
SesK seskey-in

datatype *msg* =
  Num nat |
  Agent agent-id |
  Pwd agent-id |
  Key key |
  Mult key-id key-id (**infixl** ‹⊗› *70*) |
  Hash msg |
  Crypt key msg |
  MPair msg msg |
  IDInfo agent-id msg |
  Log msg

syntax
  -MPair :: [′*a*, *args*] ⇒ ′*a* ∗ ′*b* (‹(*2*⦃-,/ -⦄)›)
  -IDInfo :: [*agent-id*, *msg*] ⇒ *msg*   (‹(*2*⟨-,/ -⟩)›)
syntax-consts
  -MPair ⇌ *MPair* **and**
  -IDInfo ⇌ *IDInfo*
translations
  ⦃*X*, *Y*, *Z*⦄ ⇌ ⦃*X*, ⦃*Y*, *Z*⦄⦄
  ⦃*X*, *Y*⦄ ⇌ *CONST MPair X Y*
  ⟨*n*, *X*⟩ ⇌ *CONST IDInfo n X*

abbreviation *SigKey* :: *msg* **where**
SigKey ≡ *Key SigK*

abbreviation *VerKey* :: *msg* **where**
VerKey ≡ *Key VerK*

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

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

abbreviation *ShaKey* :: *key-id* ⇒ *msg* **where**
ShaKey ≡ *Key* ∘ *ShaK*

abbreviation *SesKey* :: *seskey-in* ⇒ *msg* **where**
SesKey ≡ *Key* ∘ *SesK*

primrec *InvK* :: *key* ⇒ *key* **where**
InvK SigK = *VerK* |

InvK VerK = SigK |
InvK (PriK A) = PubK A |
InvK (PubK A) = PriK A |
InvK (ShaK SK) = ShaK SK |
InvK (SesK SK) = SesK SK

abbreviation *InvKey :: key ⇒ msg* **where**
InvKey ≡ Key ∘ 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

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

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

crypts-hash [*intro*]:
  foldr Crypt KS (Hash X) ∈ crypts H ⟹ foldr Crypt KS X ∈ crypts H |

crypts-fst [*intro*]:
  foldr Crypt KS ⦃X, Y⦄ ∈ crypts H ⟹ foldr Crypt KS X ∈ crypts H |

crypts-snd [*intro*]:
  foldr Crypt KS ⦃X, Y⦄ ∈ crypts H ⟹ foldr Crypt KS Y ∈ crypts H

definition *key-sets :: msg ⇒ msg set ⇒ msg set set* **where**
key-sets X H ≡ {InvKey ' set KS | KS. foldr Crypt KS X ∈ H}

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

definition *crypts-msg :: msg ⇒ msg set* **where**

crypts-msg X ≡ crypts {X}

definition *key-sets-msg :: msg ⇒ msg ⇒ msg set set* **where**
key-sets-msg X Y ≡ key-sets X {Y}

fun *seskey-set :: seskey-in ⇒ key-id set* **where**
seskey-set (Some S, U, V) = insert S (U ∪ V) |
seskey-set (None, U, V) = U ∪ V

definition *Auth-PriK :: agent-id ⇒ key-id* **where**
Auth-PriK ≡ SOME f. infinite (− range f)

abbreviation *Auth-PriKey :: agent-id ⇒ msg* **where**
Auth-PriKey ≡ PriKey ∘ Auth-PriK

abbreviation *Auth-PubKey :: agent-id ⇒ msg* **where**
Auth-PubKey ≡ PubKey ∘ Auth-PriK

consts *Auth-ShaK :: agent-id ⇒ key-id*

abbreviation *Auth-ShaKey :: agent-id ⇒ key* **where**
Auth-ShaKey ≡ ShaK ∘ Auth-ShaK

abbreviation *Sign :: agent-id ⇒ key-id ⇒ msg* **where**
Sign n A ≡ Crypt SigK {|Hash (Agent n), Hash (PubKey A)|}

abbreviation *Token :: agent-id ⇒ key-id ⇒ key-id ⇒ key-id ⇒ seskey-in ⇒ msg*
where *Token n A B C SK ≡ {|Crypt (SesK SK) (PubKey C),*
  Crypt (SesK SK) (A ⊗ B), Crypt (SesK SK) (Sign n A)|}

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

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

consts *bad-shak :: key-id set*

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

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

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

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

definition *bad-prik :: key-id set* **where**
bad-prik ≡ SOME U. U ⊆ range Auth-PriK

abbreviation *bad-prikey* :: *agent-id set* **where**
bad-prikey ≡ *Auth-PriK* −' *bad-prik*

abbreviation *bad-shakey* :: *agent-id set* **where**
bad-shakey ≡ *Auth-ShaK* −' *bad-shak*

abbreviation *bad-id-password* :: *agent-id set* **where**
bad-id-password ≡ *bad-id-pwd* ∩ *bad-pwd*

abbreviation *bad-id-prikey* :: *agent-id set* **where**
bad-id-prikey ≡ (*bad-agent* ∪ *bad-id-pubk* ∪ *bad-id-prik*) ∩ *bad-prikey*

abbreviation *bad-id-pubkey* :: *agent-id set* **where**
bad-id-pubkey ≡ *bad-agent* ∪ *bad-id-pubk* ∪ *bad-id-prik* ∩ *bad-prikey*

abbreviation *bad-id-shakey* :: *agent-id set* **where**
bad-id-shakey ≡ *bad-id-shak* ∩ *bad-shakey*

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 ≡ *range* (λ*n*. (*Asset n*, *Auth-PriKey n*)) ∪ {*Spy*} × *insert VerKey*
  (*range Num* ∪ *range Auth-PubKey* ∪ *range* (λ*n*. *Sign n* (*Auth-PriK n*)) ∪
  Agent ' *bad-agent* ∪ *Pwd* ' *bad-pwd* ∪ *PriKey* ' *bad-prik* ∪ *ShaKey* ' *bad-shak* ∪
  (λ*n*. ⟨*n*, *Pwd n*⟩) ' *bad-id-password* ∪
  (λ*n*. ⟨*n*, *Auth-PriKey n*⟩) ' *bad-id-prikey* ∪
  (λ*n*. ⟨*n*, *Auth-PubKey n*⟩) ' *bad-id-pubkey* ∪
  (λ*n*. ⟨*n*, *Key* (*Auth-ShaKey n*)⟩) ' *bad-id-shakey*)

abbreviation *rel-asset-i* :: (*state* × *state*) *set* **where**
rel-asset-i ≡ {(*s*, *s'*) | *s s'* *n S*.
  s' = *insert* (*Asset n*, *PriKey S*) *s* ∪
    {*Asset n*, *Spy*} × {*Crypt* (*Auth-ShaKey n*) (*PriKey S*)} ∪
    {(*Spy*, *Log* (*Crypt* (*Auth-ShaKey n*) (*PriKey S*)))} ∧
  PriKey S ∉ *used s*}

abbreviation *rel-owner-ii* :: (*state* × *state*) *set* **where**
rel-owner-ii ≡ {(*s*, *s'*) | *s s'* *n S A K*.
  s' = *insert* (*Owner n*, *PriKey A*) *s* ∪

17

$\{Owner\ n,\ Spy\} \times \{\!\{Num\ 1,\ PubKey\ A\}\!\} \cup$
$\quad \{Spy\} \times Log\ `\ \{Crypt\ K\ (PriKey\ S),\ \{\!\{Num\ 1,\ PubKey\ A\}\!\}\} \wedge$
$Crypt\ K\ (PriKey\ S) \in used\ s\ \wedge$
$PriKey\ A \notin used\ s\}$

abbreviation *rel-asset-ii* :: (*state* × *state*) *set* **where**
rel-asset-ii ≡ $\{(s,\ s') \mid s\ s'\ n\ A\ B.$
$\quad s' = insert\ (Asset\ n,\ PriKey\ B)\ s\ \cup$
$\quad\quad \{Asset\ n,\ Spy\} \times \{\!\{Num\ 2,\ PubKey\ B\}\!\} \cup$
$\quad\quad \{Spy\} \times Log\ `\ \{\{\!\{Num\ 1,\ PubKey\ A\}\!\},\ \{\!\{Num\ 2,\ PubKey\ B\}\!\}\} \wedge$
$\quad \{\!\{Num\ 1,\ PubKey\ A\}\!\} \in used\ s\ \wedge$
$\quad PriKey\ B \notin used\ s\}$

abbreviation *rel-owner-iii* :: (*state* × *state*) *set* **where**
rel-owner-iii ≡ $\{(s,\ s') \mid s\ s'\ n\ B\ C.$
$\quad s' = insert\ (Owner\ n,\ PriKey\ C)\ s\ \cup$
$\quad\quad \{Owner\ n,\ Spy\} \times \{\!\{Num\ 3,\ PubKey\ C\}\!\} \cup$
$\quad\quad \{Spy\} \times Log\ `\ \{\{\!\{Num\ 2,\ PubKey\ B\}\!\},\ \{\!\{Num\ 3,\ PubKey\ C\}\!\}\} \wedge$
$\quad \{\!\{Num\ 2,\ PubKey\ B\}\!\} \in used\ s\ \wedge$
$\quad PriKey\ C \notin used\ s\}$

abbreviation *rel-asset-iii* :: (*state* × *state*) *set* **where**
rel-asset-iii ≡ $\{(s,\ s') \mid s\ s'\ n\ C\ D.$
$\quad s' = insert\ (Asset\ n,\ PriKey\ D)\ s\ \cup$
$\quad\quad \{Asset\ n,\ Spy\} \times \{\!\{Num\ 4,\ PubKey\ D\}\!\} \cup$
$\quad\quad \{Spy\} \times Log\ `\ \{\{\!\{Num\ 3,\ PubKey\ C\}\!\},\ \{\!\{Num\ 4,\ PubKey\ D\}\!\}\} \wedge$
$\quad \{\!\{Num\ 3,\ PubKey\ C\}\!\} \in used\ s\ \wedge$
$\quad PriKey\ D \notin used\ s\}$

abbreviation *rel-owner-iv* :: (*state* × *state*) *set* **where**
rel-owner-iv ≡ $\{(s,\ s') \mid s\ s'\ n\ S\ A\ B\ C\ D\ K\ SK.$
$\quad s' = insert\ (Owner\ n,\ SesKey\ SK)\ s\ \cup$
$\quad\quad \{Owner\ n,\ Spy\} \times \{Crypt\ (SesK\ SK)\ (PubKey\ D)\} \cup$
$\quad\quad \{Spy\} \times Log\ `\ \{\{\!\{Num\ 4,\ PubKey\ D\}\!\},\ Crypt\ (SesK\ SK)\ (PubKey\ D)\} \wedge$
$\quad \{Crypt\ K\ (PriKey\ S),\ \{\!\{Num\ 2,\ PubKey\ B\}\!\},\ \{\!\{Num\ 4,\ PubKey\ D\}\!\}\} \subseteq used\ s\ \wedge$
$\quad \{Owner\ n\} \times \{\{\!\{Num\ 1,\ PubKey\ A\}\!\},\ \{\!\{Num\ 3,\ PubKey\ C\}\!\}\} \subseteq s\ \wedge$
$\quad SK = (if\ K = Auth\text{-}ShaKey\ n\ then\ Some\ S\ else\ None,\ \{A,\ B\},\ \{C,\ D\})\}$

abbreviation *rel-asset-iv* :: (*state* × *state*) *set* **where**
rel-asset-iv ≡ $\{(s,\ s') \mid s\ s'\ n\ S\ A\ B\ C\ D\ SK.$
$\quad s' = s\ \cup \{Asset\ n\} \times \{SesKey\ SK,\ PubKey\ B\} \cup$
$\quad\quad \{Asset\ n,\ Spy\} \times \{Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK\} \cup$
$\quad\quad \{Spy\} \times Log\ `\ \{Crypt\ (SesK\ SK)\ (PubKey\ D),$
$\quad\quad\quad Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK\} \wedge$
$\quad \{Asset\ n\} \times \{Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S),$
$\quad\quad \{\!\{Num\ 2,\ PubKey\ B\}\!\},\ \{\!\{Num\ 4,\ PubKey\ D\}\!\}\} \subseteq s\ \wedge$
$\quad \{\{\!\{Num\ 1,\ PubKey\ A\}\!\},\ \{\!\{Num\ 3,\ PubKey\ C\}\!\},$
$\quad\quad Crypt\ (SesK\ SK)\ (PubKey\ D)\} \subseteq used\ s\ \wedge$
$\quad (Asset\ n,\ PubKey\ B) \notin s\ \wedge$

$SK = (Some\ S,\ \{A,\ B\},\ \{C,\ D\})\}$

abbreviation *rel-owner-v* :: (*state* × *state*) *set* **where**
rel-owner-v ≡ {(*s*, *s′*) | *s s′ n A B C SK*.
  s′ = *s* ∪ {*Owner n*, *Spy*} × {*Crypt* (*SesK SK*) (*Pwd n*)} ∪
    {*Spy*} × *Log* ' {*Token n A B C SK*, *Crypt* (*SesK SK*) (*Pwd n*)} ∧
  Token n A B C SK ∈ *used s* ∧
  (*Owner n*, *SesKey SK*) ∈ *s* ∧
  B ∈ *fst* (*snd SK*)}

abbreviation *rel-asset-v* :: (*state* × *state*) *set* **where**
rel-asset-v ≡ {(*s*, *s′*) | *s s′ n SK*.
  s′ = *s* ∪ {*Asset n*, *Spy*} × {*Crypt* (*SesK SK*) (*Num 0*)} ∪
    {*Spy*} × *Log* ' {*Crypt* (*SesK SK*) (*Pwd n*), *Crypt* (*SesK SK*) (*Num 0*)} ∧
  (*Asset n*, *SesKey SK*) ∈ *s* ∧
  Crypt (*SesK SK*) (*Pwd n*) ∈ *used s*}

abbreviation *rel-prik* :: (*state* × *state*) *set* **where**
rel-prik ≡ {(*s*, *s′*) | *s s′ A*.
  s′ = *insert* (*Spy*, *PriKey A*) *s* ∧
  PriKey A ∉ *used s*}

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

abbreviation *rel-sesk* :: (*state* × *state*) *set* **where**
rel-sesk ≡ {(*s*, *s′*) | *s s′ A B C D S*.
  s′ = *insert* (*Spy*, *SesKey* (*Some S*, {*A*, *B*}, {*C*, *D*})) *s* ∧
  {*PriKey S*, *PriKey A*, *PubKey B*, *PriKey C*, *PubKey D*} ⊆ *spied s*}

abbreviation *rel-fact* :: (*state* × *state*) *set* **where**
rel-fact ≡ {(*s*, *s′*) | *s s′ A B*.
  s′ = *s* ∪ {*Spy*} × {*PriKey A*, *PriKey B*} ∧
  A ⊗ *B* ∈ *spied s* ∧
  (*PriKey A* ∈ *spied s* ∨ *PriKey B* ∈ *spied s*)}

abbreviation *rel-mult* :: (*state* × *state*) *set* **where**
rel-mult ≡ {(*s*, *s′*) | *s s′ A B*.
  s′ = *insert* (*Spy*, *A* ⊗ *B*) *s* ∧
  {*PriKey A*, *PriKey B*} ⊆ *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*, *Key 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-agent* :: (*state* × *state*) *set* **where**
rel-id-agent ≡ {(*s*, *s*′) | *s s*′ *n*.
  s′ = *insert* (*Spy*, ⟨*n*, *Agent n*⟩) *s* ∧
  Agent n ∈ *spied s*}

abbreviation *rel-id-invk* :: (*state* × *state*) *set* **where**
rel-id-invk ≡ {(*s*, *s*′) | *s s*′ *n K*.
  s′ = *insert* (*Spy*, ⟨*n*, *InvKey K*⟩) *s* ∧
  {*InvKey K*, ⟨*n*, *Key K*⟩} ⊆ *spied s*}

abbreviation *rel-id-sesk* :: (*state* × *state*) *set* **where**
rel-id-sesk ≡ {(*s*, *s*′) | *s s*′ *n A SK X U*.
  s′ = *s* ∪ {*Spy*} × {⟨*n*, *PubKey A*⟩, ⟨*n*, *SesKey SK*⟩} ∧
  {*PubKey A*, *SesKey SK*} ⊆ *spied s* ∧
  (⟨*n*, *PubKey A*⟩ ∈ *spied s* ∨ ⟨*n*, *SesKey SK*⟩ ∈ *spied s*) ∧
  A ∈ *seskey-set SK* ∧
  SesKey SK ∈ *U* ∧
  U ∈ *key-sets X* (*crypts* (*Log* −' *spied s*))}

abbreviation *rel-id-fact* :: (*state* × *state*) *set* **where**
rel-id-fact ≡ {(*s*, *s*′) | *s s*′ *n A B*.
  s′ = *s* ∪ {*Spy*} × {⟨*n*, *PriKey A*⟩, ⟨*n*, *PriKey B*⟩} ∧
  {*PriKey A*, *PriKey B*, ⟨*n*, *A* ⊗ *B*⟩} ⊆ *spied s*}

abbreviation *rel-id-mult* :: (*state* × *state*) *set* **where**
rel-id-mult ≡ {(*s*, *s*′) | *s s*′ *n A B U*.
  s′ = *insert* (*Spy*, ⟨*n*, *A* ⊗ *B*⟩) *s* ∧
  U ∪ {*PriKey A*, *PriKey B*, *A* ⊗ *B*} ⊆ *spied s* ∧

$(\langle n,\ PriKey\ A\rangle \in spied\ s \lor \langle n,\ PriKey\ B\rangle \in spied\ s)\ \land$
$U \in key\text{-}sets\ (A \otimes B)\ (crypts\ (Log\ -\text{`}\ spied\ s))\}$

abbreviation *rel-id-hash* :: (*state* $\times$ *state*) *set* **where**
rel-id-hash $\equiv \{(s,\ s') \mid s\ s'\ n\ X\ U.$
  $s' = s \cup \{Spy\} \times \{\langle n,\ X\rangle,\ \langle n,\ Hash\ X\rangle\}\ \land$
  $U \cup \{X,\ Hash\ X\} \subseteq spied\ s\ \land$
  $(\langle n,\ X\rangle \in spied\ s \lor \langle n,\ Hash\ X\rangle \in spied\ s)\ \land$
  $U \in key\text{-}sets\ (Hash\ X)\ (crypts\ (Log\ -\text{`}\ spied\ s))\}$

abbreviation *rel-id-crypt* :: (*state* $\times$ *state*) *set* **where**
rel-id-crypt $\equiv \{(s,\ s') \mid s\ s'\ n\ X\ U.$
  $s' = s \cup \{Spy\} \times IDInfo\ n\ \text{`}\ insert\ X\ U\ \land$
  $insert\ X\ U \subseteq spied\ s\ \land$
  $(\langle n,\ X\rangle \in spied\ s \lor (\exists\ K \in U.\ \langle n,\ K\rangle \in spied\ s))\ \land$
  $U \in key\text{-}sets\ X\ (crypts\ (Log\ -\text{`}\ spied\ s))\}$

abbreviation *rel-id-sep* :: (*state* $\times$ *state*) *set* **where**
rel-id-sep $\equiv \{(s,\ s') \mid s\ s'\ n\ X\ Y.$
  $s' = s \cup \{Spy\} \times \{\langle n,\ X\rangle,\ \langle n,\ Y\rangle\}\ \land$
  $\{X,\ Y,\ \langle n,\ \{\!| X,\ Y |\!\}\rangle\} \subseteq spied\ s\}$

abbreviation *rel-id-con* :: (*state* $\times$ *state*) *set* **where**
rel-id-con $\equiv \{(s,\ s') \mid s\ s'\ n\ X\ Y\ U.$
  $s' = insert\ (Spy,\ \langle n,\ \{\!| X,\ Y |\!\}\rangle)\ s\ \land$
  $U \cup \{X,\ Y,\ \{\!| X,\ Y |\!\}\} \subseteq spied\ s\ \land$
  $(\langle n,\ X\rangle \in spied\ s \lor \langle n,\ Y\rangle \in spied\ s)\ \land$
  $U \in key\text{-}sets\ \{\!| X,\ Y |\!\}\ (crypts\ (Log\ -\text{`}\ spied\ s))\}$

definition *rel* :: (*state* $\times$ *state*) *set* **where**
rel $\equiv$ *rel-asset-i* $\cup$ *rel-owner-ii* $\cup$ *rel-asset-ii* $\cup$ *rel-owner-iii* $\cup$
  rel-asset-iii $\cup$ *rel-owner-iv* $\cup$ *rel-asset-iv* $\cup$ *rel-owner-v* $\cup$ *rel-asset-v* $\cup$
  rel-prik $\cup$ *rel-pubk* $\cup$ *rel-sesk* $\cup$ *rel-fact* $\cup$ *rel-mult* $\cup$ *rel-hash* $\cup$ *rel-dec* $\cup$
  rel-enc $\cup$ *rel-sep* $\cup$ *rel-con* $\cup$ *rel-id-agent* $\cup$ *rel-id-invk* $\cup$ *rel-id-sesk* $\cup$
  rel-id-fact $\cup$ *rel-id-mult* $\cup$ *rel-id-hash* $\cup$ *rel-id-crypt* $\cup$ *rel-id-sep* $\cup$ *rel-id-con*

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   Confidentiality and authenticity properties

theory *Authentication*
 imports *Definitions*

begin

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** - $\Longrightarrow$ - $\longrightarrow$ - $\longrightarrow$ ($\exists\ u\ v.\ ?Q\ x\ y\ u\ v$))
$\langle proof \rangle$

proposition *state-subset*:
  $s \models s' \Longrightarrow s \subseteq s'$
$\langle proof \rangle$

proposition *spied-subset*:
  $s \models s' \Longrightarrow spied\ s \subseteq spied\ s'$
$\langle proof \rangle$

proposition *used-subset*:
  $s \models s' \Longrightarrow used\ s \subseteq used\ s'$
$\langle proof \rangle$

proposition *asset-ii-init*:
  $[\![s_0 \models s;\ (Asset\ n,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s]\!] \Longrightarrow$
    $PriKey\ A \notin spied\ s_0$
$\langle proof \rangle$

proposition *auth-prikey-used*:
  $s_0 \models s \Longrightarrow Auth\text{-}PriKey\ n \in used\ s$
$\langle proof \rangle$

proposition *asset-i-used*:
  $s_0 \models s \Longrightarrow$
    $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ A)) \in s \longrightarrow$
  $PriKey\ A \in used\ s$
$\langle proof \rangle$

proposition *owner-ii-used*:
  $s_0 \models s \Longrightarrow$
    $(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \longrightarrow$
  $PriKey\ A \in used\ s$
$\langle proof \rangle$

proposition *asset-ii-used*:
  $s_0 \models s \Longrightarrow$
    $(Asset\ n,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
  $PriKey\ A \in used\ s$
$\langle proof \rangle$

proposition *owner-iii-used*:
  $s_0 \models s \Longrightarrow$

$(Owner\ n,\ \{\!|Num\ 3,\ PubKey\ A|\!\}) \in s \longrightarrow$
  $PriKey\ A \in used\ s$
$\langle proof \rangle$

proposition *asset-iii-used*:
  $s_0 \models s \Longrightarrow$
    $(Asset\ n,\ \{\!|Num\ 4,\ PubKey\ A|\!\}) \in s \longrightarrow$
  $PriKey\ A \in used\ s$
$\langle proof \rangle$

proposition *asset-i-unique* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    $(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (PriKey\ A)) \in s \longrightarrow$
    $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ A)) \in s \longrightarrow$
  $m = n$
$\langle proof \rangle$

proposition *owner-ii-unique* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    $(Owner\ m,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \longrightarrow$
    $(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \longrightarrow$
  $m = n$
$\langle proof \rangle$

proposition *asset-ii-unique* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    $(Asset\ m,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
    $(Asset\ n,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
  $m = n$
$\langle proof \rangle$

proposition *auth-prikey-asset-i* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    $(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (Auth\text{-}PriKey\ n)) \in s \longrightarrow$
  $False$
$\langle proof \rangle$

proposition *auth-pubkey-owner-ii* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    $(Owner\ m,\ \{\!|Num\ 1,\ Auth\text{-}PubKey\ n|\!\}) \in s \longrightarrow$
  $False$
$\langle proof \rangle$

proposition *auth-pubkey-owner-iii* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    $(Owner\ m,\ \{\!|Num\ 3,\ Auth\text{-}PubKey\ n|\!\}) \in s \longrightarrow$
  $False$
$\langle proof \rangle$

proposition *auth-pubkey-asset-ii* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (*Asset m*, {|*Num 2*, *Auth-PubKey n*|}) $\in s \longrightarrow$
 False
$\langle proof \rangle$

proposition *auth-pubkey-asset-iii* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (*Asset m*, {|*Num 4*, *Auth-PubKey n*|}) $\in s \longrightarrow$
 False
$\langle proof \rangle$

proposition *asset-i-owner-ii* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (*Asset m*, *Crypt* (*Auth-ShaKey m*) (*PriKey A*)) $\in s \longrightarrow$
   (*Owner n*, {|*Num 1*, *PubKey A*|}) $\in s \longrightarrow$
 False
$\langle proof \rangle$

proposition *asset-i-owner-iii* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (*Asset m*, *Crypt* (*Auth-ShaKey m*) (*PriKey A*)) $\in s \longrightarrow$
   (*Owner n*, {|*Num 3*, *PubKey A*|}) $\in s \longrightarrow$
 False
$\langle proof \rangle$

proposition *asset-i-asset-ii* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (*Asset m*, *Crypt* (*Auth-ShaKey m*) (*PriKey A*)) $\in s \longrightarrow$
   (*Asset n*, {|*Num 2*, *PubKey A*|}) $\in s \longrightarrow$
 False
$\langle proof \rangle$

proposition *asset-i-asset-iii* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (*Asset m*, *Crypt* (*Auth-ShaKey m*) (*PriKey A*)) $\in s \longrightarrow$
   (*Asset n*, {|*Num 4*, *PubKey A*|}) $\in s \longrightarrow$
 False
$\langle proof \rangle$

proposition *asset-ii-owner-ii* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (*Asset m*, {|*Num 2*, *PubKey A*|}) $\in s \longrightarrow$
   (*Owner n*, {|*Num 1*, *PubKey A*|}) $\in s \longrightarrow$
 False
$\langle proof \rangle$

proposition *asset-ii-owner-iii* [*rule-format*]:
 $s_0 \models s \Longrightarrow$

$(Asset\ m,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
$(Owner\ n,\ \{\!|Num\ 3,\ PubKey\ A|\!\}) \in s \longrightarrow$
$\ \ False$
$\langle proof \rangle$

proposition *asset-ii-asset-iii* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\ \ (Asset\ m,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
$\ \ (Asset\ n,\ \{\!|Num\ 4,\ PubKey\ A|\!\}) \in s \longrightarrow$
$\ \ False$
$\langle proof \rangle$

proposition *asset-iii-owner-iii* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\ \ (Asset\ m,\ \{\!|Num\ 4,\ PubKey\ A|\!\}) \in s \longrightarrow$
$\ \ (Owner\ n,\ \{\!|Num\ 3,\ PubKey\ A|\!\}) \in s \longrightarrow$
$\ \ False$
$\langle proof \rangle$

proposition *asset-iv-state* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\ \ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s \longrightarrow$
$\ \ (\exists\,A\ D.\ fst\ (snd\ SK) = \{A,\ B\} \wedge snd\ (snd\ SK) = \{C,\ D\} \wedge$
$\ \ \ \ (Asset\ n,\ \{\!|Num\ 2,\ PubKey\ B|\!\}) \in s \wedge (Asset\ n,\ \{\!|Num\ 4,\ PubKey\ D|\!\}) \in s \wedge$
$\ \ \ \ Crypt\ (SesK\ SK)\ (PubKey\ D) \in used\ s \wedge (Asset\ n,\ PubKey\ B) \in s)$
$\langle proof \rangle$

proposition *owner-v-state* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\ \ (Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s \longrightarrow$
$\ (Owner\ n,\ SesKey\ SK) \in s \wedge$
$\ (\exists\,A\ B\ C.\ Token\ n\ A\ B\ C\ SK \in used\ s \wedge B \in fst\ (snd\ SK))$
$\langle proof \rangle$

proposition *asset-v-state* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\ \ (Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0)) \in s \longrightarrow$
$\ (Asset\ n,\ SesKey\ SK) \in s \wedge Crypt\ (SesK\ SK)\ (Pwd\ n) \in used\ s$
$\langle proof \rangle$

lemma *owner-seskey-nonce-1*:
$[\![\,s \vdash s';$
$\ \ (Owner\ n,\ SesKey\ SK) \in s \longrightarrow$
$\ \ \ (\exists\,S.\ fst\ SK = Some\ S \wedge Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S) \in used\ s) \vee$
$\ \ \ fst\ SK = None;$
$\ \ (Owner\ n,\ SesKey\ SK) \in s'\,]\!] \Longrightarrow$
$\ (\exists\,S.\ fst\ SK = Some\ S \wedge Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S) \in used\ s') \vee$
$\ fst\ SK = None$
$\langle proof \rangle$

proposition *owner-seskey-nonce* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (Owner n, SesKey SK) $\in s \longrightarrow$
 $(\exists S.\ \mathit{fst}\ SK = \mathit{Some}\ S \wedge \mathit{Crypt}\ (\mathit{Auth\text{-}ShaKey}\ n)\ (\mathit{PriKey}\ S) \in \mathit{used}\ s) \vee$
 fst SK = None
⟨*proof*⟩

proposition *owner-seskey-other* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (Owner n, SesKey SK) $\in s \longrightarrow$
 $(\exists A\ B\ C\ D.\ \mathit{fst}\ (\mathit{snd}\ SK) = \{A,\ B\} \wedge \mathit{snd}\ (\mathit{snd}\ SK) = \{C,\ D\} \wedge$
   (Owner n, ⦃Num 1, PubKey A⦄) $\in s \wedge$
   (Owner n, ⦃Num 3, PubKey C⦄) $\in s \wedge$
   (Owner n, Crypt (SesK SK) (PubKey D)) $\in s)$
⟨*proof*⟩

proposition *asset-seskey-nonce* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (Asset n, SesKey SK) $\in s \longrightarrow$
 $(\exists S.\ \mathit{fst}\ SK = \mathit{Some}\ S \wedge (\mathit{Asset}\ n,\ \mathit{Crypt}\ (\mathit{Auth\text{-}ShaKey}\ n)\ (\mathit{PriKey}\ S)) \in s)$
⟨*proof*⟩

proposition *asset-seskey-other* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
   (Asset n, SesKey SK) $\in s \longrightarrow$
 $(\exists A\ B\ C\ D.\ \mathit{fst}\ (\mathit{snd}\ SK) = \{A,\ B\} \wedge \mathit{snd}\ (\mathit{snd}\ SK) = \{C,\ D\} \wedge$
   (Asset n, ⦃Num 2, PubKey B⦄) $\in s \wedge (\mathit{Asset}\ n,\ ⦃\mathit{Num}\ 4,\ \mathit{PubKey}\ D⦄) \in s \wedge$
   (Asset n, Token n (Auth-PriK n) B C SK) $\in s)$
⟨*proof*⟩

declare *Range-Un-eq* [*simp*]

proposition *used-prod* [*simp*]:
 $A \neq \{\} \Longrightarrow \mathit{used}\ (A \times H) = H$
⟨*proof*⟩

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

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

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

proposition *parts-msg-num* [*simp*]:
 parts-msg (*Num n*) = {*Num n*}
⟨*proof*⟩

proposition *parts-msg-pwd* [*simp*]:
 parts-msg (*Pwd n*) = {*Pwd n*}
⟨*proof*⟩

proposition *parts-msg-key* [*simp*]:
 parts-msg (*Key K*) = {*Key K*}
⟨*proof*⟩

proposition *parts-msg-mult* [*simp*]:
 parts-msg $(A \otimes B) = \{A \otimes B\}$
⟨*proof*⟩

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

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

lemma *parts-crypt-2*:
 insert (*Crypt K X*) (*parts* {*X*}) $\subseteq$ *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-trace* [*simp*]:
  parts-msg (*Log X*) = {*Log X*}
⟨*proof*⟩

proposition *parts-idinfo* [*simp*]:
  parts (*IDInfo n ' H*) = *IDInfo n ' H*
⟨*proof*⟩

proposition *parts-trace* [*simp*]:
  parts (*Log ' H*) = *Log ' H*
⟨*proof*⟩

proposition *parts-dec*:
  $\llbracket$ s′ = *insert* (*Spy, X*) s ∧ (*Spy, Crypt K X*) ∈ s ∧ (*Spy, Key* (*InvK K*)) ∈ s;
    Y ∈ *parts-msg X* $\rrbracket$ ⟹
  Y ∈ *parts* (*used s*)
⟨*proof*⟩

proposition *parts-enc*:
  $\llbracket$ s′ = *insert* (*Spy, Crypt K X*) s ∧ (*Spy, X*) ∈ s ∧ (*Spy, Key K*) ∈ s;
    Y ∈ *parts-msg X* $\rrbracket$ ⟹
  Y ∈ *parts* (*used s*)
⟨*proof*⟩

proposition *parts-sep*:
  $\llbracket$ s′ = *insert* (*Spy, X*) (*insert* (*Spy, Y*) s) ∧ (*Spy,* {|X, Y|}) ∈ s;
    Z ∈ *parts-msg X* ∨ Z ∈ *parts-msg Y* $\rrbracket$ ⟹
  Z ∈ *parts* (*used s*)
⟨*proof*⟩

proposition *parts-con*:
  $\llbracket$ s′ = *insert* (*Spy,* {|X, Y|}) s ∧ (*Spy, X*) ∈ s ∧ (*Spy, Y*) ∈ s;
    Z ∈ *parts-msg X* ∨ Z ∈ *parts-msg Y* $\rrbracket$ ⟹

$Z \in parts\ (used\ s)$
$\langle proof \rangle$

lemma *parts-init-1*:
 $parts\ (used\ s_0) \subseteq used\ s_0 \cup range\ (Hash \circ Agent) \cup$
   $range\ (Hash \circ Auth\text{-}PubKey) \cup$
   $range\ (\lambda n.\ \{\!|\,Hash\ (Agent\ n),\ Hash\ (Auth\text{-}PubKey\ n)\,\!|\})$
$\langle proof \rangle$

lemma *parts-init-2*:
 $used\ s_0 \cup range\ (Hash \circ Agent) \cup range\ (Hash \circ Auth\text{-}PubKey) \cup$
   $range\ (\lambda n.\ \{\!|\,Hash\ (Agent\ n),\ Hash\ (Auth\text{-}PubKey\ n)\,\!|\}) \subseteq parts\ (used\ s_0)$
$\langle proof \rangle$

proposition *parts-init*:
 $parts\ (used\ s_0) = used\ s_0 \cup range\ (Hash \circ Agent) \cup$
   $range\ (Hash \circ Auth\text{-}PubKey) \cup$
   $range\ (\lambda n.\ \{\!|\,Hash\ (Agent\ n),\ Hash\ (Auth\text{-}PubKey\ n)\,\!|\})$
$\langle proof \rangle$

proposition *parts-crypt-prikey-start*:
 $[\![\,s \vdash s';\ Crypt\ K\ (PriKey\ A) \in parts\ (used\ s');$
   $Crypt\ K\ (PriKey\ A) \notin parts\ (used\ s)\,]\!] \Longrightarrow$
 $(\exists n.\ K = Auth\text{-}ShaKey\ n\ \wedge$
   $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ A)) \in s')\ \vee$
 $\{PriKey\ A,\ Key\ K\} \subseteq spied\ s'$
$\langle proof \rangle$

proposition *parts-crypt-prikey*:
 $[\![\,s_0 \models s;\ Crypt\ K\ (PriKey\ A) \in parts\ (used\ s)\,]\!] \Longrightarrow$
   $(\exists n.\ K = Auth\text{-}ShaKey\ n\ \wedge$
     $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ A)) \in s)\ \vee$
   $\{PriKey\ A,\ Key\ K\} \subseteq spied\ s$
$\langle proof \rangle$

proposition *parts-crypt-pubkey-start*:
 $[\![\,s \vdash s';\ Crypt\ (SesK\ SK)\ (PubKey\ C) \in parts\ (used\ s');$
   $Crypt\ (SesK\ SK)\ (PubKey\ C) \notin parts\ (used\ s)\,]\!] \Longrightarrow$
 $C \in snd\ (snd\ SK)\ \wedge\ ((\exists n.\ (Owner\ n,\ SesKey\ SK) \in s')\ \vee$
   $(\exists n\ B.\ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s'))\ \vee$
 $SesKey\ SK \in spied\ s'$
$\langle proof \rangle$

proposition *parts-crypt-pubkey*:
 $[\![\,s_0 \models s;\ Crypt\ (SesK\ SK)\ (PubKey\ C) \in parts\ (used\ s)\,]\!] \Longrightarrow$
   $C \in snd\ (snd\ SK)\ \wedge\ ((\exists n.\ (Owner\ n,\ SesKey\ SK) \in s)\ \vee$
     $(\exists n\ B.\ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s))\ \vee$

    SesKey SK ∈ spied s
⟨*proof*⟩

proposition *parts-crypt-key-start*:
  ⟦*s ⊢ s′; Crypt K (Key K′) ∈ parts (used s′);*
    Crypt K (Key K′) ∉ parts (used s); K′ ∉ range PriK ∪ range PubK⟧ ⟹
  {*Key K′, Key K*} ⊆ *spied s′*
⟨*proof*⟩

proposition *parts-crypt-key*:
  ⟦*s₀ ⊨ s; Crypt K (Key K′) ∈ parts (used s);*
    K′ ∉ range PriK ∪ range PubK⟧ ⟹
  {*Key K′, Key K*} ⊆ *spied s*
⟨*proof*⟩

proposition *parts-crypt-sign-start*:
  ⟦*s ⊢ s′; Crypt (SesK SK) (Sign n A) ∈ parts (used s′);*
    Crypt (SesK SK) (Sign n A) ∉ parts (used s)⟧ ⟹
  (*Asset n, SesKey SK) ∈ s′ ∨ SesKey SK ∈ spied s′*
⟨*proof*⟩

proposition *parts-crypt-sign*:
  ⟦*s₀ ⊨ s; Crypt (SesK SK) (Sign n A) ∈ parts (used s)*⟧ ⟹
  (*Asset n, SesKey SK) ∈ s ∨ SesKey SK ∈ spied s*
⟨*proof*⟩

proposition *parts-crypt-pwd-start*:
  ⟦*s ⊢ s′; Crypt K (Pwd n) ∈ parts (used s′);*
    Crypt K (Pwd n) ∉ parts (used s)⟧ ⟹
  (∃ *SK. K = SesK SK ∧ (Owner n, Crypt (SesK SK) (Pwd n)) ∈ s′) ∨*
  {*Pwd n, Key K*} ⊆ *spied s′*
⟨*proof*⟩

proposition *parts-crypt-pwd*:
  ⟦*s₀ ⊨ s; Crypt K (Pwd n) ∈ parts (used s)*⟧ ⟹
  (∃ *SK. K = SesK SK ∧ (Owner n, Crypt (SesK SK) (Pwd n)) ∈ s) ∨*
  {*Pwd n, Key K*} ⊆ *spied s*
⟨*proof*⟩

proposition *parts-crypt-num-start*:
  ⟦*s ⊢ s′; Crypt (SesK SK) (Num 0) ∈ parts (used s′);*
    Crypt (SesK SK) (Num 0) ∉ parts (used s)⟧ ⟹
  (∃ *n. (Asset n, Crypt (SesK SK) (Num 0)) ∈ s′) ∨ SesKey SK ∈ spied s′*
⟨*proof*⟩

proposition *parts-crypt-num*:
$\llbracket s_0 \models s;\ Crypt\ (SesK\ SK)\ (Num\ 0) \in parts\ (used\ s) \rrbracket \Longrightarrow$
  $(\exists\, n.\ (Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0)) \in s) \lor SesKey\ SK \in spied\ s$
$\langle proof \rangle$

proposition *parts-crypt-mult-start*:
$\llbracket s \vdash s';\ Crypt\ (SesK\ SK)\ (A \otimes B) \in parts\ (used\ s');$
  $Crypt\ (SesK\ SK)\ (A \otimes B) \notin parts\ (used\ s) \rrbracket \Longrightarrow$
$B \in fst\ (snd\ SK) \land (\exists\, n\ C.\ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s') \lor$
$\{A \otimes B,\ SesKey\ SK\} \subseteq spied\ s$
$\langle proof \rangle$

proposition *parts-crypt-mult*:
$\llbracket s_0 \models s;\ Crypt\ (SesK\ SK)\ (A \otimes B) \in parts\ (used\ s) \rrbracket \Longrightarrow$
  $B \in fst\ (snd\ SK) \land (\exists\, n\ C.\ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s) \lor$
  $\{A \otimes B,\ SesKey\ SK\} \subseteq spied\ s$
$\langle proof \rangle$

proposition *parts-mult-start*:
$\llbracket s \vdash s';\ A \otimes B \in parts\ (used\ s');\ A \otimes B \notin parts\ (used\ s) \rrbracket \Longrightarrow$
  $(\exists\, n\ SK.\ A = Auth\text{-}PriK\ n \land (Asset\ n,\ \{\!|Num\ 2,\ PubKey\ B|\!\}) \in s' \land$
    $Crypt\ (SesK\ SK)\ (A \otimes B) \in parts\ (used\ s')) \lor$
  $\{PriKey\ A,\ PriKey\ B\} \subseteq spied\ s'$
$\langle proof \rangle$

proposition *parts-mult*:
$\llbracket s_0 \models s;\ A \otimes B \in parts\ (used\ s) \rrbracket \Longrightarrow$
  $(\exists\, n.\ A = Auth\text{-}PriK\ n \land (Asset\ n,\ \{\!|Num\ 2,\ PubKey\ B|\!\}) \in s) \lor$
  $\{PriKey\ A,\ PriKey\ B\} \subseteq spied\ s$
$\langle proof \rangle$

proposition *parts-mpair-key-start*:
$\llbracket s \vdash s';\ \{\!|X,\ Y|\!\} \in parts\ (used\ s');\ \{\!|X,\ Y|\!\} \notin parts\ (used\ s);$
  $X = Key\ K \lor Y = Key\ K \land K \notin range\ PubK \rrbracket \Longrightarrow$
  $\{X,\ Y\} \subseteq spied\ s'$
$\langle proof \rangle$

proposition *parts-mpair-key*:
$\llbracket s_0 \models s;\ \{\!|X,\ Y|\!\} \in parts\ (used\ s);$
  $X = Key\ K \lor Y = Key\ K \land K \notin range\ PubK \rrbracket \Longrightarrow$
  $\{X,\ Y\} \subseteq spied\ s$
$\langle proof \rangle$

proposition *parts-mpair-pwd-start*:
$\llbracket s \vdash s';\ \{\!|X,\ Y|\!\} \in parts\ (used\ s');\ \{\!|X,\ Y|\!\} \notin parts\ (used\ s);$

$X = Pwd\ n\ \lor\ Y = Pwd\ n\rrbracket \Longrightarrow$
  $\{X,\ Y\} \subseteq spied\ s'$
$\langle proof \rangle$

proposition *parts-mpair-pwd*:
$\llbracket s_0 \models s;\ \{\!|X,\ Y|\!\} \in parts\ (used\ s);\ X = Pwd\ n\ \lor\ Y = Pwd\ n\rrbracket \Longrightarrow$
  $\{X,\ Y\} \subseteq spied\ s$
$\langle proof \rangle$

proposition *parts-pubkey-false-start*:
  assumes
    A: $s_0 \models s$ **and**
    B: $s \vdash s'$ **and**
    C: $Crypt\ (SesK\ SK)\ (PubKey\ C) \in parts\ (used\ s')$ **and**
    D: $Crypt\ (SesK\ SK)\ (PubKey\ C) \notin parts\ (used\ s)$ **and**
    E: $\forall\, n.\ (Owner\ n,\ SesKey\ SK) \notin s'$ **and**
    F: $SesKey\ SK \notin spied\ s'$
  shows *False*
$\langle proof \rangle$

proposition *parts-pubkey-false*:
$\llbracket s_0 \models s;\ Crypt\ (SesK\ SK)\ (PubKey\ C) \in parts\ (used\ s);$
  $\forall\, n.\ (Owner\ n,\ SesKey\ SK) \notin s;\ SesKey\ SK \notin spied\ s\rrbracket \Longrightarrow$
  False
$\langle proof \rangle$

proposition *asset-ii-spied-start*:
  assumes
    A: $s_0 \models s$ **and**
    B: $s \vdash s'$ **and**
    C: $PriKey\ B \in spied\ s'$ **and**
    D: $PriKey\ B \notin spied\ s$ **and**
    E: $(Asset\ n,\ \{\!|Num\ 2,\ PubKey\ B|\!\}) \in s$
  shows $Auth\text{-}PriKey\ n \in spied\ s\ \land$
    $(\exists\, C\ SK.\ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s)$
    (**is** $- \land (\exists\, C\ SK.\ ?P\ n\ C\ SK\ s)$)
$\langle proof \rangle$

proposition *asset-ii-spied*:
  assumes
    A: $s_0 \models s$ **and**
    B: $PriKey\ B \in spied\ s$ **and**
    C: $(Asset\ n,\ \{\!|Num\ 2,\ PubKey\ B|\!\}) \in s$
  shows $Auth\text{-}PriKey\ n \in spied\ s\ \land$
    $(\exists\, C\ SK.\ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s)$
    (**is** $?P\ s$)
$\langle proof \rangle$

proposition *asset-iv-unique*:
  assumes
    A: $s_0 \models s$ **and**
    B: (*Asset m, Token m (Auth-PriK m) B C$'$ SK$'$*) $\in s$ **and**
    C: (*Asset n, Token n (Auth-PriK n) B C SK*) $\in s$
     (**is** *?P n C SK s*)
  shows $m = n \land C' = C \land SK' = SK$
$\langle proof \rangle$

theorem *sigkey-secret*:
 $s_0 \models s \implies SigKey \notin spied\ s$
$\langle proof \rangle$

proposition *parts-sign-start*:
  assumes *A*: $s_0 \models s$
  shows $[\![s \vdash s'; Sign\ n\ A \in parts\ (used\ s'); Sign\ n\ A \notin parts\ (used\ s)]\!] \implies$
   $A = Auth\text{-}PriK\ n$
$\langle proof \rangle$

proposition *parts-sign*:
 $[\![s_0 \models s; Sign\ n\ A \in parts\ (used\ s)]\!] \implies$
   $A = Auth\text{-}PriK\ n$
$\langle proof \rangle$

theorem *auth-shakey-secret*:
 $[\![s_0 \models s; n \notin bad\text{-}shakey]\!] \implies$
   Key (Auth-ShaKey n) $\notin spied\ s$
$\langle proof \rangle$

theorem *auth-prikey-secret*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin bad\text{-}prikey$
  shows *Auth-PriKey n* $\notin spied\ s$
$\langle proof \rangle$

proposition *asset-ii-secret*:
 $[\![s_0 \models s; n \notin bad\text{-}prikey; (Asset\ n, \{\!|Num\ 2, PubKey\ B|\!\}) \in s]\!] \implies$
   PriKey B $\notin spied\ s$
$\langle proof \rangle$

proposition *asset-i-secret* [*rule-format*]:
  assumes

33

    A: $s_0 \models s$ **and**
    B: $n \notin$ *bad-shakey*
  shows (*Asset n*, *Crypt* (*Auth-ShaKey n*) (*PriKey S*)) $\in s \longrightarrow$
    PriKey S $\notin$ *spied s*
⟨*proof*⟩

proposition *owner-ii-secret* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    (*Owner n*, ⦃*Num 1*, *PubKey A*⦄) $\in s \longrightarrow$
  PriKey A $\notin$ *spied s*
⟨*proof*⟩

proposition *seskey-spied* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    SesKey SK $\in$ *spied s* $\longrightarrow$
  ($\exists$ *S A C*. *fst SK* = *Some S* $\wedge$ *A* $\in$ *fst* (*snd SK*) $\wedge$ *C* $\in$ *snd* (*snd SK*) $\wedge$
    {*PriKey S*, *PriKey A*, *PriKey C*} $\subseteq$ *spied s*)
  (**is** - $\Longrightarrow$ - $\longrightarrow$ ($\exists$ *S A C*. *?P S A C s*))
⟨*proof*⟩

proposition *owner-seskey-shakey*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin$ *bad-shakey* **and**
    C: (*Owner n*, *SesKey SK*) $\in s$
  shows *SesKey SK* $\notin$ *spied s*
⟨*proof*⟩

proposition *owner-seskey-prikey*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin$ *bad-prikey* **and**
    C: (*Owner m*, *SesKey SK*) $\in s$ **and**
    D: (*Asset n*, ⦃*Num 2*, *PubKey B*⦄) $\in s$ **and**
    E: *B* $\in$ *fst* (*snd SK*)
  shows *SesKey SK* $\notin$ *spied s*
⟨*proof*⟩

proposition *asset-seskey-shakey*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin$ *bad-shakey* **and**
    C: (*Asset n*, *SesKey SK*) $\in s$
  shows *SesKey SK* $\notin$ *spied s*
⟨*proof*⟩

theorem *owner-seskey-unique*:
  assumes

     A: $s_0 \models s$ **and**
     B: (*Owner m, Crypt* (*SesK SK*) (*Pwd m*)) $\in s$ **and**
     C: (*Owner n, Crypt* (*SesK SK*) (*Pwd n*)) $\in s$
   shows $m = n$
$\langle proof \rangle$

theorem *owner-seskey-secret*:
  assumes
     A: $s_0 \models s$ **and**
     B: $n \notin$ *bad-shakey* $\cap$ *bad-prikey* **and**
     C: (*Owner n, Crypt* (*SesK SK*) (*Pwd n*)) $\in s$
   shows *SesKey SK* $\notin$ *spied s*
$\langle proof \rangle$

theorem *owner-num-genuine*:
  assumes
     A: $s_0 \models s$ **and**
     B: $n \notin$ *bad-shakey* $\cap$ *bad-prikey* **and**
     C: (*Owner n, Crypt* (*SesK SK*) (*Pwd n*)) $\in s$ **and**
     D: *Crypt* (*SesK SK*) (*Num 0*) $\in$ *used s*
   shows (*Asset n, Crypt* (*SesK SK*) (*Num 0*)) $\in s$
$\langle proof \rangle$

theorem *owner-token-genuine*:
  assumes
     A: $s_0 \models s$ **and**
     B: $n \notin$ *bad-shakey* $\cap$ *bad-prikey* **and**
     C: (*Owner n,* ⦃*Num 3, PubKey C*⦄) $\in s$ **and**
     D: (*Owner n, Crypt* (*SesK SK*) (*Pwd n*)) $\in s$ **and**
     E: *Token n A B C SK* $\in$ *used s*
   shows $A = Auth\text{-}PriK\ n \wedge B \in fst\ (snd\ SK) \wedge C \in snd\ (snd\ SK) \wedge$
     (*Asset n,* ⦃*Num 2, PubKey B*⦄) $\in s \wedge$ (*Asset n, Token n A B C SK*) $\in s$
     (**is** *?P n A* $\wedge$ *?Q B* $\wedge$ *?R C* $\wedge$ *?S n B* $\wedge$ -)
$\langle proof \rangle$

theorem *pwd-secret*:
  assumes
     A: $s_0 \models s$ **and**
     B: $n \notin$ *bad-pwd* $\cup$ *bad-shakey* $\cap$ *bad-prikey*
   shows *Pwd n* $\notin$ *spied s*
$\langle proof \rangle$

theorem *asset-seskey-unique*:
  assumes

    A: $s_0 \models s$ **and**
    B: (*Asset m*, *Token m* (*Auth-PriK m*) *B′ C′ SK*) ∈ *s* **and**
    C: (*Asset n*, *Token n* (*Auth-PriK n*) *B C SK*) ∈ *s*
      (**is** *?P n B C SK s*)
  shows *m = n ∧ B′ = B ∧ C′ = C*
⟨*proof*⟩

theorem *asset-seskey-secret*:
 assumes
   A: $s_0 \models s$ **and**
   B: *n* ∉ *bad-shakey* ∩ (*bad-pwd* ∪ *bad-prikey*) **and**
   C: (*Asset n*, *Crypt* (*SesK SK*) (*Num 0*)) ∈ *s*
  shows *SesKey SK* ∉ *spied s*
⟨*proof*⟩

theorem *asset-pwd-genuine*:
 assumes
   A: $s_0 \models s$ **and**
   B: *n* ∉ *bad-shakey* ∩ (*bad-pwd* ∪ *bad-prikey*) **and**
   C: (*Asset n*, *Crypt* (*SesK SK*) (*Num 0*)) ∈ *s*
  shows (*Owner n*, *Crypt* (*SesK SK*) (*Pwd n*)) ∈ *s*
⟨*proof*⟩

theorem *asset-token-genuine*:
 assumes
   A: $s_0 \models s$ **and**
   B: *n* ∉ *bad-shakey* ∩ (*bad-pwd* ∪ *bad-prikey*) **and**
   C: (*Asset n*, ⦃*Num 4*, *PubKey D*⦄) ∈ *s* **and**
   D: (*Asset n*, *Crypt* (*SesK SK*) (*Num 0*)) ∈ *s* **and**
   E: *D* ∈ *snd* (*snd SK*)
  shows (*Owner n*, *Crypt* (*SesK SK*) (*PubKey D*)) ∈ *s*
⟨*proof*⟩

proposition *owner-iii-secret* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
  (*Owner n*, ⦃*Num 3*, *PubKey C*⦄) ∈ *s* ⟶
 PriKey C ∉ *spied s*
⟨*proof*⟩

proposition *asset-iii-secret* [*rule-format*]:
 $s_0 \models s \Longrightarrow$
  (*Asset n*, ⦃*Num 4*, *PubKey D*⦄) ∈ *s* ⟶
 PriKey D ∉ *spied s*
⟨*proof*⟩

theorem *seskey-forward-secret*:
  assumes
    A: $s_0 \models s$ **and**
     B: (*Owner m*, *Crypt* (*SesK SK*) (*Pwd m*)) $\in s$ **and**
     C: (*Asset n*, *Crypt* (*SesK SK*) (*Num 0*)) $\in s$
  shows $m = n \land SesKey\ SK \notin spied\ s$
$\langle proof \rangle$

end

3   Anonymity properties

theory *Anonymity*
  imports *Authentication*
begin

proposition *crypts-empty* [*simp*]:
 crypts {} = {}
$\langle proof \rangle$

proposition *crypts-mono*:
 $H \subseteq H' \Longrightarrow crypts\ H \subseteq crypts\ H'$
$\langle proof \rangle$

lemma *crypts-union-1*:
 crypts $(H \cup H') \subseteq crypts\ H \cup crypts\ H'$
$\langle proof \rangle$

lemma *crypts-union-2*:
 crypts $H \cup crypts\ H' \subseteq crypts\ (H \cup H')$
$\langle proof \rangle$

proposition *crypts-union*:
 crypts $(H \cup H') = crypts\ H \cup crypts\ H'$
$\langle proof \rangle$

proposition *crypts-insert*:
 crypts (*insert X H*) = *crypts-msg X* $\cup$ *crypts H*
$\langle proof \rangle$

proposition *crypts-msg-num* [*simp*]:
 crypts-msg (*Num n*) = {*Num n*}
$\langle proof \rangle$

proposition *crypts-msg-agent* [*simp*]:
 crypts-msg (*Agent n*) = {*Agent n*}
$\langle proof \rangle$

proposition *crypts-msg-pwd* [*simp*]:
 crypts-msg (*Pwd n*) = {*Pwd n*}
⟨*proof*⟩

proposition *crypts-msg-key* [*simp*]:
 crypts-msg (*Key K*) = {*Key K*}
⟨*proof*⟩

proposition *crypts-msg-mult* [*simp*]:
 crypts-msg (*A* ⊗ *B*) = {*A* ⊗ *B*}
⟨*proof*⟩

lemma *crypts-hash-1*:
 crypts {*Hash X*} ⊆ *insert* (*Hash X*) (*crypts* {*X*})
⟨*proof*⟩

lemma *crypts-hash-2*:
 insert (*Hash X*) (*crypts* {*X*}) ⊆ *crypts* {*Hash X*}
⟨*proof*⟩

proposition *crypts-msg-hash* [*simp*]:
 crypts-msg (*Hash X*) = *insert* (*Hash X*) (*crypts-msg X*)
⟨*proof*⟩

proposition *crypts-comp*:
 X ∈ *crypts H* ⟹ *Crypt K X* ∈ *crypts* (*Crypt K* ' *H*)
⟨*proof*⟩

lemma *crypts-crypt-1*:
 crypts {*Crypt K X*} ⊆ *Crypt K* ' *crypts* {*X*}
⟨*proof*⟩

lemma *crypts-crypt-2*:
 Crypt K ' *crypts* {*X*} ⊆ *crypts* {*Crypt K X*}
⟨*proof*⟩

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

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

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

proposition *crypts-msg-mpair* [*simp*]:

crypts-msg $\{X, Y\}$ = *insert* $\{X, Y\}$ (*crypts-msg X* $\cup$ *crypts-msg Y*)

⟨*proof*⟩

proposition *foldr-crypt-size*:

 size (*foldr Crypt KS X*) = *size X* + *length KS*

⟨*proof*⟩

proposition *key-sets-empty* [*simp*]:

 key-sets X {} = {}

⟨*proof*⟩

proposition *key-sets-mono*:

 $H \subseteq H' \Longrightarrow$ *key-sets X H* $\subseteq$ *key-sets X H'*

⟨*proof*⟩

proposition *key-sets-union*:

 key-sets X ($H \cup H'$) = *key-sets X H* $\cup$ *key-sets X H'*

⟨*proof*⟩

proposition *key-sets-insert*:

 key-sets X (*insert Y H*) = *key-sets-msg X Y* $\cup$ *key-sets X H*

⟨*proof*⟩

proposition *key-sets-msg-eq*:

 key-sets-msg X X = {{}}

⟨*proof*⟩

proposition *key-sets-msg-num* [*simp*]:

 key-sets-msg X (*Num n*) = (*if X* = *Num n then* {{}} *else* {})

⟨*proof*⟩

proposition *key-sets-msg-agent* [*simp*]:

 key-sets-msg X (*Agent n*) = (*if X* = *Agent n then* {{}} *else* {})

⟨*proof*⟩

proposition *key-sets-msg-pwd* [*simp*]:

 key-sets-msg X (*Pwd n*) = (*if X* = *Pwd n then* {{}} *else* {})

⟨*proof*⟩

proposition *key-sets-msg-key* [*simp*]:

 key-sets-msg X (*Key K*) = (*if X* = *Key K then* {{}} *else* {})

⟨*proof*⟩

proposition *key-sets-msg-mult* [*simp*]:

 key-sets-msg X ($A \otimes B$) = (*if X* = $A \otimes B$ *then* {{}} *else* {})

⟨*proof*⟩

proposition *key-sets-msg-hash* [*simp*]:

key-sets-msg X (Hash Y) = (if X = Hash Y then {{}} else {})
⟨*proof*⟩

lemma *key-sets-crypt-1*:
  X ≠ Crypt K Y ⟹
    key-sets X {Crypt K Y} ⊆ insert (InvKey K) ' key-sets X {Y}
⟨*proof*⟩

lemma *key-sets-crypt-2*:
  insert (InvKey K) ' key-sets X {Y} ⊆ key-sets X {Crypt K Y}
⟨*proof*⟩

proposition *key-sets-msg-crypt* [*simp*]:
  key-sets-msg X (Crypt K Y) = (if X = Crypt K Y then {{}} else
    insert (InvKey K) ' key-sets-msg X Y)
⟨*proof*⟩

proposition *key-sets-msg-mpair* [*simp*]:
  key-sets-msg X ⦃Y, Z⦄ = (if X = ⦃Y, Z⦄ then {{}} else {})
⟨*proof*⟩

proposition *key-sets-range*:
  U ∈ key-sets X H ⟹ U ⊆ range Key
⟨*proof*⟩

proposition *key-sets-crypts-hash*:
  key-sets (Hash X) (crypts H) ⊆ key-sets X (crypts H)
⟨*proof*⟩

proposition *key-sets-crypts-fst*:
  key-sets ⦃X, Y⦄ (crypts H) ⊆ key-sets X (crypts H)
⟨*proof*⟩

proposition *key-sets-crypts-snd*:
  key-sets ⦃X, Y⦄ (crypts H) ⊆ key-sets Y (crypts H)
⟨*proof*⟩

lemma *log-spied-1*:
  ⟦*s ⊢ s′;*
    Log X ∈ parts (used s) ⟶ Log X ∈ spied s;
    Log X ∈ parts (used s′)⟧ ⟹
  Log X ∈ spied s′
⟨*proof*⟩

proposition *log-spied* [*rule-format*]:
  s₀ ⊨ s ⟹
    Log X ∈ parts (used s) ⟶
  Log X ∈ spied s

⟨*proof*⟩

proposition *log-dec*:
  ⟦$s_0 \models s$; $s' = insert\ (Spy,\ X)\ s \wedge (Spy,\ Crypt\ K\ X) \in s\ \wedge$
    $(Spy,\ Key\ (InvK\ K)) \in s$⟧ $\Longrightarrow$
  key-sets Y (*crypts* $\{Y.\ Log\ Y = X\}$) $\subseteq$ *key-sets* Y (*crypts* ($Log\ -`\ spied\ s$))
⟨*proof*⟩

proposition *log-sep*:
  ⟦$s_0 \models s$; $s' = insert\ (Spy,\ X)\ (insert\ (Spy,\ Y)\ s) \wedge (Spy,\ \{\!|X,\ Y|\!\}) \in s$⟧ $\Longrightarrow$
    key-sets Z (*crypts* $\{Z.\ Log\ Z = X\}$) $\subseteq$ *key-sets* Z (*crypts* ($Log\ -`\ spied\ s$)) $\wedge$
    key-sets Z (*crypts* $\{Z.\ Log\ Z = Y\}$) $\subseteq$ *key-sets* Z (*crypts* ($Log\ -`\ spied\ s$))
⟨*proof*⟩

lemma *idinfo-spied-1*:
  ⟦$s \vdash s'$;
    $\langle n,\ X \rangle \in parts\ (used\ s) \longrightarrow \langle n,\ X \rangle \in spied\ s$;
    $\langle n,\ X \rangle \in parts\ (used\ s')$⟧ $\Longrightarrow$
  $\langle n,\ X \rangle \in spied\ s'$
⟨*proof*⟩

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

proposition *idinfo-dec*:
  ⟦$s_0 \models s$; $s' = insert\ (Spy,\ X)\ s \wedge (Spy,\ Crypt\ K\ X) \in s\ \wedge$
    $(Spy,\ Key\ (InvK\ K)) \in s$; $\langle n,\ Y \rangle = X$⟧ $\Longrightarrow$
  $\langle n,\ Y \rangle \in spied\ s$
⟨*proof*⟩

proposition *idinfo-sep*:
  ⟦$s_0 \models s$; $s' = insert\ (Spy,\ X)\ (insert\ (Spy,\ Y)\ s) \wedge (Spy,\ \{\!|X,\ Y|\!\}) \in s$;
    $\langle n,\ Z \rangle = X \vee \langle n,\ Z \rangle = Y$⟧ $\Longrightarrow$
  $\langle n,\ Z \rangle \in spied\ s$
⟨*proof*⟩

lemma *idinfo-msg-1*:
  assumes A: $s_0 \models s$
  shows ⟦$s \vdash s'$; $\langle n,\ X \rangle \in spied\ s \longrightarrow X \in spied\ s$; $\langle n,\ X \rangle \in spied\ s'$⟧ $\Longrightarrow$
    $X \in spied\ s'$
⟨*proof*⟩

proposition *idinfo-msg* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    $\langle n, X \rangle \in spied\ s \longrightarrow$
  $X \in spied\ s$
$\langle proof \rangle$

proposition *parts-agent-start*:
  $\llbracket s \vdash s';\ Agent\ n \in parts\ (used\ s');\ Agent\ n \notin parts\ (used\ s) \rrbracket \Longrightarrow False$
$\langle proof \rangle$

proposition *parts-agent* [*rotated*]:
  assumes A: $n \notin bad\text{-}agent$
  shows $s_0 \models s \Longrightarrow Agent\ n \notin parts\ (used\ s)$
$\langle proof \rangle$

lemma *idinfo-init-1* [*rule-format*]:
  assumes A: $s_0 \models s$
  shows $\llbracket s \vdash s';\ n \notin bad\text{-}id\text{-}password \cup bad\text{-}id\text{-}pubkey \cup bad\text{-}id\text{-}shakey;$
    $\forall X.\ \langle n, X \rangle \notin spied\ s \rrbracket \Longrightarrow$
  $\langle n, X \rangle \notin spied\ s'$
$\langle proof \rangle$

proposition *idinfo-init*:
  $\llbracket s_0 \models s;\ n \notin bad\text{-}id\text{-}password \cup bad\text{-}id\text{-}pubkey \cup bad\text{-}id\text{-}shakey \rrbracket \Longrightarrow$
  $\langle n, X \rangle \notin spied\ s$
$\langle proof \rangle$

lemma *idinfo-mpair-1* [*rule-format*]:
  $\llbracket (s, s') \in rel\text{-}id\text{-}hash \cup rel\text{-}id\text{-}crypt \cup rel\text{-}id\text{-}sep \cup rel\text{-}id\text{-}con;$
    $\forall X\ Y.\ \langle n, \{\!|X,\ Y|\!\} \rangle \in spied\ s \longrightarrow$
      $key\text{-}sets\ \{\!|X,\ Y|\!\}\ (crypts\ (Log\ -\text{`}\ spied\ s)) \neq \{\};$
    $\langle n, \{\!|X,\ Y|\!\} \rangle \in spied\ s' \rrbracket \Longrightarrow$
  $key\text{-}sets\ \{\!|X,\ Y|\!\}\ (crypts\ (Log\ -\text{`}\ spied\ s')) \neq \{\}$
$\langle proof \rangle$

lemma *idinfo-mpair-2* [*rule-format*]:
  assumes A: $s_0 \models s$
  shows $\llbracket s \vdash s';\ (s, s') \notin rel\text{-}id\text{-}hash \cup rel\text{-}id\text{-}crypt \cup rel\text{-}id\text{-}sep \cup rel\text{-}id\text{-}con;$
    $\forall X\ Y.\ \langle n, \{\!|X,\ Y|\!\} \rangle \in spied\ s \longrightarrow$
      $key\text{-}sets\ \{\!|X,\ Y|\!\}\ (crypts\ (Log\ -\text{`}\ spied\ s)) \neq \{\};$
    $\langle n, \{\!|X,\ Y|\!\} \rangle \in spied\ s' \rrbracket \Longrightarrow$
  $key\text{-}sets\ \{\!|X,\ Y|\!\}\ (crypts\ (Log\ -\text{`}\ spied\ s')) \neq \{\}$
$\langle proof \rangle$

proposition *idinfo-mpair* [*rule-format*]:
  $s_0 \models s \Longrightarrow$

$\langle n, \{X, Y\} \rangle \in spied\ s \longrightarrow$
  $key\text{-}sets\ \{X, Y\}\ (crypts\ (Log\ -\text{'}\ spied\ s)) \neq \{\}$
$\langle proof \rangle$

proposition *key-sets-pwd-empty*:
  $s_0 \models s \Longrightarrow$
    $key\text{-}sets\ (Hash\ (Pwd\ n))\ (crypts\ (Log\ -\text{'}\ spied\ s)) = \{\}\ \wedge$
    $key\text{-}sets\ \{Pwd\ n, X\}\ (crypts\ (Log\ -\text{'}\ spied\ s)) = \{\}\ \wedge$
    $key\text{-}sets\ \{X, Pwd\ n\}\ (crypts\ (Log\ -\text{'}\ spied\ s)) = \{\}$
  (**is** - $\Longrightarrow$ *key-sets ?X (?H s)* = - $\wedge$ *key-sets ?Y* - = - $\wedge$ *key-sets ?Z* - = -)
$\langle proof \rangle$

proposition *key-sets-pwd-seskey* [*rule-format*]:
  $s_0 \models s \Longrightarrow$
    $U \in key\text{-}sets\ (Pwd\ n)\ (crypts\ (Log\ -\text{'}\ spied\ s)) \longrightarrow$
  $(\exists SK.\ U = \{SesKey\ SK\}\ \wedge$
    $((Owner\ n, Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s\ \vee$
    $(Asset\ n, Crypt\ (SesK\ SK)\ (Num\ 0)) \in s))$
  (**is** - $\Longrightarrow$ - $\longrightarrow$ *?P s*)
$\langle proof \rangle$

lemma *pwd-anonymous-1* [*rule-format*]:
  $[\![s_0 \models s;\ n \notin bad\text{-}id\text{-}password]\!] \Longrightarrow$
    $\langle n, Pwd\ n \rangle \in spied\ s \longrightarrow$
  $(\exists SK.\ SesKey\ SK \in spied\ s\ \wedge$
    $((Owner\ n, Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s\ \vee$
    $(Asset\ n, Crypt\ (SesK\ SK)\ (Num\ 0)) \in s))$
  (**is** $[\![-;\ -]\!] \Longrightarrow$ - $\longrightarrow$ *?P s*)
$\langle proof \rangle$

theorem *pwd-anonymous*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin bad\text{-}id\text{-}password$ **and**
    C: $n \notin bad\text{-}shakey \cap (bad\text{-}pwd \cup bad\text{-}prikey) \cap (bad\text{-}id\text{-}pubkey \cup bad\text{-}id\text{-}shak)$
  shows $\langle n, Pwd\ n \rangle \notin spied\ s$
$\langle proof \rangle$

proposition *idinfo-pwd-start*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin bad\text{-}agent$
  shows $[\![s \vdash s';\ \exists X.\ \langle n, X \rangle \in spied\ s'\ \wedge\ X \neq Pwd\ n;$
    $\neg\ (\exists X.\ \langle n, X \rangle \in spied\ s\ \wedge\ X \neq Pwd\ n)]\!] \Longrightarrow$
      $\exists SK.\ SesKey\ SK \in spied\ s\ \wedge$
        $((Owner\ n, Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s\ \vee$

$(Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0)) \in s)$
$\langle proof \rangle$

proposition *idinfo-pwd*:
$\llbracket s_0 \models s;\ \exists\, X.\ \langle n,\ X \rangle \in spied\ s \land X \neq Pwd\ n;$
$\quad n \notin bad\text{-}id\text{-}pubkey \cup bad\text{-}id\text{-}shakey \rrbracket \Longrightarrow$
$\exists\, SK.\ SesKey\ SK \in spied\ s\ \land$
$\quad ((Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s\ \lor$
$\quad (Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0)) \in s)$
$\langle proof \rangle$

theorem *auth-prikey-anonymous*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin bad\text{-}id\text{-}prikey$ **and**
    C: $n \notin bad\text{-}shakey \cap bad\text{-}prikey \cap (bad\text{-}id\text{-}password \cup bad\text{-}id\text{-}shak)$
  shows $\langle n,\ Auth\text{-}PriKey\ n \rangle \notin spied\ s$
$\langle proof \rangle$

theorem *auth-shakey-anonymous*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin bad\text{-}id\text{-}shakey$ **and**
    C: $n \notin bad\text{-}shakey \cap (bad\text{-}id\text{-}password \cup bad\text{-}id\text{-}pubkey)$
  shows $\langle n,\ Key\ (Auth\text{-}ShaKey\ n) \rangle \notin spied\ s$
$\langle proof \rangle$

end

4   Possibility properties

theory *Possibility*
  imports *Anonymity*
begin

type-synonym *seskey-tuple* $=$ *key-id* $\times$ *key-id* $\times$ *key-id* $\times$ *key-id* $\times$ *key-id*

type-synonym *stage* $=$ *state* $\times$ *seskey-tuple*

abbreviation *pred-asset-i* :: *agent-id* $\Rightarrow$ *state* $\Rightarrow$ *stage* $\Rightarrow$ *bool* **where**
pred-asset-i $n\ s\ x \equiv$
  $\exists\, S.\ PriKey\ S \notin used\ s \land x = (insert\ (Asset\ n,\ PriKey\ S)\ s\ \cup$
    $\{Asset\ n,\ Spy\} \times \{Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)\}\ \cup$
    $\{(Spy,\ Log\ (Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)))\},$
    $S,\ 0,\ 0,\ 0,\ 0)$

definition *run-asset-i* :: *agent-id* $\Rightarrow$ *state* $\Rightarrow$ *stage* **where**
run-asset-i n s $\equiv$ *SOME x. pred-asset-i n s x*

abbreviation *pred-owner-ii* :: *agent-id* $\Rightarrow$ *stage* $\Rightarrow$ *stage* $\Rightarrow$ *bool* **where**
pred-owner-ii n x y $\equiv$ *case x of (s, S, -)* $\Rightarrow$
  $\exists A.$ *PriKey A* $\notin$ *used s* $\wedge$ *y = (insert (Owner n, PriKey A) s* $\cup$
    {Owner n, Spy} $\times$ *{⦃Num 1, PubKey A⦄}* $\cup$
    {Spy} $\times$ *Log ' {Crypt (Auth-ShaKey n) (PriKey S), ⦃Num 1, PubKey A⦄},*
    S, A, 0, 0, 0)

definition *run-owner-ii* :: *agent-id* $\Rightarrow$ *state* $\Rightarrow$ *stage* **where**
run-owner-ii n s $\equiv$ *SOME x. pred-owner-ii n (run-asset-i n s) x*

abbreviation *pred-asset-ii* :: *agent-id* $\Rightarrow$ *stage* $\Rightarrow$ *stage* $\Rightarrow$ *bool* **where**
pred-asset-ii n x y $\equiv$ *case x of (s, S, A, -)* $\Rightarrow$
  $\exists B.$ *PriKey B* $\notin$ *used s* $\wedge$ *y = (insert (Asset n, PriKey B) s* $\cup$
    {Asset n, Spy} $\times$ *{⦃Num 2, PubKey B⦄}* $\cup$
    {Spy} $\times$ *Log ' {⦃Num 1, PubKey A⦄, ⦃Num 2, PubKey B⦄},*
    S, A, B, 0, 0)

definition *run-asset-ii* :: *agent-id* $\Rightarrow$ *state* $\Rightarrow$ *stage* **where**
run-asset-ii n s $\equiv$ *SOME x. pred-asset-ii n (run-owner-ii n s) x*

abbreviation *pred-owner-iii* :: *agent-id* $\Rightarrow$ *stage* $\Rightarrow$ *stage* $\Rightarrow$ *bool* **where**
pred-owner-iii n x y $\equiv$ *case x of (s, S, A, B, -)* $\Rightarrow$
  $\exists C.$ *PriKey C* $\notin$ *used s* $\wedge$ *y = (insert (Owner n, PriKey C) s* $\cup$
    {Owner n, Spy} $\times$ *{⦃Num 3, PubKey C⦄}* $\cup$
    {Spy} $\times$ *Log ' {⦃Num 2, PubKey B⦄, ⦃Num 3, PubKey C⦄},*
    S, A, B, C, 0)

definition *run-owner-iii* :: *agent-id* $\Rightarrow$ *state* $\Rightarrow$ *stage* **where**
run-owner-iii n s $\equiv$ *SOME x. pred-owner-iii n (run-asset-ii n s) x*

abbreviation *pred-asset-iii* :: *agent-id* $\Rightarrow$ *stage* $\Rightarrow$ *stage* $\Rightarrow$ *bool* **where**
pred-asset-iii n x y $\equiv$ *case x of (s, S, A, B, C, -)* $\Rightarrow$
  $\exists D.$ *PriKey D* $\notin$ *used s* $\wedge$ *y = (insert (Asset n, PriKey D) s* $\cup$
    {Asset n, Spy} $\times$ *{⦃Num 4, PubKey D⦄}* $\cup$
    {Spy} $\times$ *Log ' {⦃Num 3, PubKey C⦄, ⦃Num 4, PubKey D⦄},*
    S, A, B, C, D)

definition *run-asset-iii* :: *agent-id* $\Rightarrow$ *state* $\Rightarrow$ *stage* **where**
run-asset-iii n s $\equiv$ *SOME x. pred-asset-iii n (run-owner-iii n s) x*

abbreviation *stage-owner-iv* :: *agent-id* $\Rightarrow$ *stage* $\Rightarrow$ *stage* **where**

stage-owner-iv n x ≡ *let (s, S, A, B, C, D) = x;*
  SK = (Some S, {A, B}, {C, D}) in
  (insert (Owner n, SesKey SK) s ∪
    {Owner n, Spy} × {Crypt (SesK SK) (PubKey D)} ∪
    {Spy} × Log ' {⦃Num 4, PubKey D⦄, Crypt (SesK SK) (PubKey D)},
    S, A, B, C, D)

definition *run-owner-iv :: agent-id ⇒ state ⇒ stage* **where**
run-owner-iv n s ≡ stage-owner-iv n (run-asset-iii n s)

abbreviation *stage-asset-iv :: agent-id ⇒ stage ⇒ stage* **where**
stage-asset-iv n x ≡ *let (s, S, A, B, C, D) = x;*
  SK = (Some S, {A, B}, {C, D}) in
  (s ∪ {Asset n} × {SesKey SK, PubKey B} ∪
    {Asset n, Spy} × {Token n (Auth-PriK n) B C SK} ∪
    {Spy} × Log ' {Crypt (SesK SK) (PubKey D),
      Token n (Auth-PriK n) B C SK},
    S, A, B, C, D)

definition *run-asset-iv :: agent-id ⇒ state ⇒ stage* **where**
run-asset-iv n s ≡ stage-asset-iv n (run-owner-iv n s)

abbreviation *stage-owner-v :: agent-id ⇒ stage ⇒ stage* **where**
stage-owner-v n x ≡ *let (s, S, A, B, C, D) = x;*
  SK = (Some S, {A, B}, {C, D}) in
  (s ∪ {Owner n, Spy} × {Crypt (SesK SK) (Pwd n)} ∪
    {Spy} × Log ' {Token n (Auth-PriK n) B C SK, Crypt (SesK SK) (Pwd n)},
    S, A, B, C, D)

definition *run-owner-v :: agent-id ⇒ state ⇒ stage* **where**
run-owner-v n s ≡ stage-owner-v n (run-asset-iv n s)

abbreviation *stage-asset-v :: agent-id ⇒ stage ⇒ stage* **where**
stage-asset-v n x ≡ *let (s, S, A, B, C, D) = x;*
  SK = (Some S, {A, B}, {C, D}) in
  (s ∪ {Asset n, Spy} × {Crypt (SesK SK) (Num 0)} ∪
    {Spy} × Log ' {Crypt (SesK SK) (Pwd n), Crypt (SesK SK) (Num 0)},
    S, A, B, C, D)

definition *run-asset-v :: agent-id ⇒ state ⇒ stage* **where**
run-asset-v n s ≡ stage-asset-v n (run-owner-v n s)

lemma *prikey-unused-1*:
 infinite {A. PriKey A ∉ used s_0}
⟨*proof*⟩

46

lemma *prikey-unused-2*:
$\llbracket s \vdash s';$ *infinite* $\{A.\ PriKey\ A \notin used\ s\}\rrbracket \Longrightarrow$
   infinite $\{A.\ PriKey\ A \notin used\ s'\}$
$\langle proof \rangle$

proposition *prikey-unused*:
$s_0 \models s \Longrightarrow \exists A.\ PriKey\ A \notin used\ s$
$\langle proof \rangle$

lemma *pubkey-unused-1*:
$\llbracket s \vdash s';$ *PubKey* $A \in parts\ (used\ s) \longrightarrow PriKey\ A \in used\ s;$
   PubKey $A \in parts\ (used\ s')\rrbracket \Longrightarrow$
  PriKey $A \in used\ s'$
$\langle proof \rangle$

proposition *pubkey-unused* [*rule-format*]:
$s_0 \models s \Longrightarrow$
   PriKey $A \notin used\ s \longrightarrow$
  PubKey $A \notin parts\ (used\ s)$
$\langle proof \rangle$

proposition *run-asset-i-ex*:
$s_0 \models s \Longrightarrow pred\text{-}asset\text{-}i\ n\ s\ (run\text{-}asset\text{-}i\ n\ s)$
$\langle proof \rangle$

proposition *run-asset-i-rel*:
$s_0 \models s \Longrightarrow s \models fst\ (run\text{-}asset\text{-}i\ n\ s)$
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
$\langle proof \rangle$

proposition *run-asset-i-msg*:
$s_0 \models s \Longrightarrow$
   case run-asset-i n s of $(s',\ S,\ \text{-}) \Rightarrow$
    $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \in s'$
$\langle proof \rangle$

proposition *run-asset-i-nonce*:
$s_0 \models s \Longrightarrow PriKey\ (fst\ (snd\ (run\text{-}asset\text{-}i\ n\ s))) \notin used\ s$
$\langle proof \rangle$

proposition *run-asset-i-unused*:
$s_0 \models s \Longrightarrow \exists A.\ PriKey\ A \notin used\ (fst\ (run\text{-}asset\text{-}i\ n\ s))$
$\langle proof \rangle$

proposition *run-owner-ii-ex*:

47

$s_0 \models s \implies$ *pred-owner-ii n* (*run-asset-i n s*) (*run-owner-ii n s*)
⟨*proof*⟩

proposition *run-owner-ii-rel*:
$s_0 \models s \implies s \models$ *fst* (*run-owner-ii n s*)
   (**is** - $\implies$ - $\models$ *?t*)
⟨*proof*⟩

proposition *run-owner-ii-msg*:
$s_0 \models s \implies$
  case run-owner-ii n s of (*s′*, *S*, *A*, -) $\Rightarrow$
    {(*Asset n*, *Crypt* (*Auth-ShaKey n*) (*PriKey S*)),
     (*Owner n*, ⦃*Num 1*, *PubKey A*⦄)} $\subseteq$ *s′*
⟨*proof*⟩

proposition *run-owner-ii-nonce*:
$s_0 \models s \implies$ *PriKey* (*fst* (*snd* (*run-owner-ii n s*))) $\notin$ *used s*
⟨*proof*⟩

proposition *run-owner-ii-unused*:
$s_0 \models s \implies \exists B$. *PriKey B* $\notin$ *used* (*fst* (*run-owner-ii n s*))
⟨*proof*⟩

proposition *run-asset-ii-ex*:
$s_0 \models s \implies$ *pred-asset-ii n* (*run-owner-ii n s*) (*run-asset-ii n s*)
⟨*proof*⟩

proposition *run-asset-ii-rel*:
$s_0 \models s \implies s \models$ *fst* (*run-asset-ii n s*)
   (**is** - $\implies$ - $\models$ *?t*)
⟨*proof*⟩

proposition *run-asset-ii-msg*:
  assumes *A*: $s_0 \models s$
  shows *case run-asset-ii n s of* (*s′*, *S*, *A*, *B*, -) $\Rightarrow$
   insert (*Owner n*, ⦃*Num 1*, *PubKey A*⦄)
    ({*Asset n*} × {*Crypt* (*Auth-ShaKey n*) (*PriKey S*),
     ⦃*Num 2*, *PubKey B*⦄}) $\subseteq$ *s′* $\wedge$
   (*Asset n*, *PubKey B*) $\notin$ *s′*
⟨*proof*⟩

proposition *run-asset-ii-nonce*:
$s_0 \models s \implies$ *PriKey* (*fst* (*snd* (*run-asset-ii n s*))) $\notin$ *used s*
⟨*proof*⟩

proposition *run-asset-ii-unused*:
$s_0 \models s \implies \exists C$. *PriKey C* $\notin$ *used* (*fst* (*run-asset-ii n s*))
⟨*proof*⟩

proposition *run-owner-iii-ex*:
  $s_0 \models s \implies$ *pred-owner-iii n* (*run-asset-ii n s*) (*run-owner-iii n s*)
⟨*proof*⟩

proposition *run-owner-iii-rel*:
  $s_0 \models s \implies s \models$ *fst* (*run-owner-iii n s*)
    (**is** - $\implies$ - $\models$ *?t*)
⟨*proof*⟩

proposition *run-owner-iii-msg*:
  $s_0 \models s \implies$
    case run-owner-iii n s of (*s′, S, A, B, C, -*) $\Rightarrow$
      {*Asset n*} × {*Crypt* (*Auth-ShaKey n*) (*PriKey S*), ⦃*Num 2, PubKey B*⦄} ∪
      {*Owner n*} × {⦃*Num 1, PubKey A*⦄, ⦃*Num 3, PubKey C*⦄} ⊆ *s′* ∧
      (*Asset n, PubKey B*) ∉ *s′*
⟨*proof*⟩

proposition *run-owner-iii-nonce*:
  $s_0 \models s \implies$ *PriKey* (*fst* (*snd* (*run-owner-iii n s*))) ∉ *used s*
⟨*proof*⟩

proposition *run-owner-iii-unused*:
  $s_0 \models s \implies \exists D.$ *PriKey D* ∉ *used* (*fst* (*run-owner-iii n s*))
⟨*proof*⟩

proposition *run-asset-iii-ex*:
  $s_0 \models s \implies$ *pred-asset-iii n* (*run-owner-iii n s*) (*run-asset-iii n s*)
⟨*proof*⟩

proposition *run-asset-iii-rel*:
  $s_0 \models s \implies s \models$ *fst* (*run-asset-iii n s*)
    (**is** - $\implies$ - $\models$ *?t*)
⟨*proof*⟩

proposition *run-asset-iii-msg*:
  $s_0 \models s \implies$
    case run-asset-iii n s of (*s′, S, A, B, C, D*) $\Rightarrow$
      {*Asset n*} × {*Crypt* (*Auth-ShaKey n*) (*PriKey S*), ⦃*Num 2, PubKey B*⦄,
        ⦃*Num 4, PubKey D*⦄} ∪
      {*Owner n*} × {⦃*Num 1, PubKey A*⦄, ⦃*Num 3, PubKey C*⦄} ⊆ *s′* ∧
      (*Asset n, PubKey B*) ∉ *s′*
⟨*proof*⟩

proposition *run-asset-iii-nonce*:
  $s_0 \models s \implies$ *PriKey* (*fst* (*snd* (*run-asset-iii n s*))) ∉ *used s*
⟨*proof*⟩

lemma *run-owner-iv-rel-1*:
$\llbracket s_0 \models s;\ run\text{-}asset\text{-}iii\ n\ s = (s',\ S,\ A,\ B,\ C,\ D) \rrbracket \implies$
$\quad s \models fst\ (run\text{-}owner\text{-}iv\ n\ s)$
$\qquad (\textbf{is}\ \llbracket \text{-};\ \text{-}\rrbracket \implies \text{-} \models\ ?t)$
$\langle proof \rangle$

proposition *run-owner-iv-rel*:
$\quad s_0 \models s \implies s \models fst\ (run\text{-}owner\text{-}iv\ n\ s)$
$\langle proof \rangle$

proposition *run-owner-iv-msg*:
$\quad s_0 \models s \implies$
$\quad\ let\ (s',\ S,\ A,\ B,\ C,\ D) = run\text{-}owner\text{-}iv\ n\ s;$
$\quad\quad SK = (Some\ S,\ \{A,\ B\},\ \{C,\ D\})\ in$
$\quad\quad \{Asset\ n\} \times \{Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S),\ \{\!|Num\ 2,\ PubKey\ B|\!\},$
$\quad\quad\quad \{\!|Num\ 4,\ PubKey\ D|\!\}\} \cup$
$\quad\quad \{Owner\ n\} \times \{\{\!|Num\ 1,\ PubKey\ A|\!\},\ \{\!|Num\ 3,\ PubKey\ C|\!\},\ SesKey\ SK,$
$\quad\quad\quad Crypt\ (SesK\ SK)\ (PubKey\ D)\} \subseteq s' \wedge$
$\quad\quad (Asset\ n,\ PubKey\ B) \notin s'$
$\langle proof \rangle$

proposition *run-owner-iv-nonce*:
$\quad s_0 \models s \implies PriKey\ (fst\ (snd\ (run\text{-}owner\text{-}iv\ n\ s))) \notin used\ s$
$\langle proof \rangle$

proposition *run-asset-iv-rel*:
$\quad s_0 \models s \implies s \models fst\ (run\text{-}asset\text{-}iv\ n\ s)$
$\quad\ (\textbf{is}\ \text{-} \implies \text{-} \models\ ?t)$
$\langle proof \rangle$

proposition *run-asset-iv-msg*:
$\quad s_0 \models s \implies$
$\quad\ let\ (s',\ S,\ A,\ B,\ C,\ D) = run\text{-}asset\text{-}iv\ n\ s;\ SK = (Some\ S,\ \{A,\ B\},\ \{C,\ D\})\ in$
$\quad\quad insert\ (Owner\ n,\ SesKey\ SK)$
$\quad\quad\quad (\{Asset\ n\} \times \{SesKey\ SK,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK\}) \subseteq s'$
$\langle proof \rangle$

proposition *run-asset-iv-nonce*:
$\quad s_0 \models s \implies PriKey\ (fst\ (snd\ (run\text{-}asset\text{-}iv\ n\ s))) \notin used\ s$
$\langle proof \rangle$

proposition *run-owner-v-rel*:
$\quad s_0 \models s \implies s \models fst\ (run\text{-}owner\text{-}v\ n\ s)$
$\quad\ (\textbf{is}\ \text{-} \implies \text{-} \models\ ?t)$
$\langle proof \rangle$

proposition *run-owner-v-msg*:
 $s_0 \models s \Longrightarrow$
   let $(s', S, A, B, C, D) = run\text{-}owner\text{-}v\ n\ s;$
     $SK = (Some\ S, \{A, B\}, \{C, D\})\ in$
     $\{(Asset\ n,\ SesKey\ SK),$
       $(Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n))\} \subseteq s'$
 $\langle proof \rangle$

proposition *run-owner-v-nonce*:
 $s_0 \models s \Longrightarrow PriKey\ (fst\ (snd\ (run\text{-}owner\text{-}v\ n\ s))) \notin used\ s$
 $\langle proof \rangle$

proposition *run-asset-v-rel*:
 $s_0 \models s \Longrightarrow s \models fst\ (run\text{-}asset\text{-}v\ n\ s)$
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
 $\langle proof \rangle$

proposition *run-asset-v-msg*:
 $s_0 \models s \Longrightarrow$
   let $(s', S, A, B, C, D) = run\text{-}asset\text{-}v\ n\ s;\ SK = (Some\ S, \{A, B\}, \{C, D\})\ in$
     $\{(Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)),$
       $(Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0))\} \subseteq s'$
 $\langle proof \rangle$

proposition *run-asset-v-nonce*:
 $s_0 \models s \Longrightarrow PriKey\ (fst\ (snd\ (run\text{-}asset\text{-}v\ n\ s))) \notin used\ s$
 $\langle proof \rangle$

lemma *runs-unbounded-1*:
 $[\![s_0 \models s;\ run\text{-}asset\text{-}v\ n\ s = (s', S, A, B, C, D)]\!] \Longrightarrow$
   $\exists s'\ S\ SK.\ (Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \notin s \wedge$
     $\{(Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)),$
       $(Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0))\} \subseteq s' \wedge$
     $s \models s' \wedge fst\ SK = Some\ S$
 $\langle proof \rangle$

theorem *runs-unbounded*:
 $s_0 \models s \Longrightarrow \exists s'\ S\ SK.\ s \models s' \wedge fst\ SK = Some\ S \wedge$
   $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \notin s \wedge$
   $\{(Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)),$
     $(Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0))\} \subseteq s'$
 $\langle proof \rangle$

definition *pwd-spy-i* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-i $n \equiv$

51

(*insert* (*Spy*, *Crypt* (*Auth-ShaKey n*) (*Auth-PriKey n*)) s_0,
  Auth-PriK n, *0*, *0*, *0*, *0*)

definition *pwd-owner-ii* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-owner-ii n $\equiv$ *SOME x. pred-owner-ii n* (*pwd-spy-i n*) *x*

definition *pwd-spy-ii* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-ii n $\equiv$
  case pwd-owner-ii n of (*s, S, A, -*) $\Rightarrow$
    (*insert* (*Spy*, $\{\!|$*Num 2*, *PubKey S*$|\!\}$) *s, S, A, S, 0, 0*)

definition *pwd-owner-iii* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-owner-iii n $\equiv$ *SOME x. pred-owner-iii n* (*pwd-spy-ii n*) *x*

definition *pwd-spy-iii* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-iii n $\equiv$
  case pwd-owner-iii n of (*s, S, A, B, C, -*) $\Rightarrow$
    (*insert* (*Spy*, $\{\!|$*Num 4*, *PubKey S*$|\!\}$) *s, S, A, B, C, S*)

definition *pwd-owner-iv* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-owner-iv n $\equiv$ *stage-owner-iv n* (*pwd-spy-iii n*)

definition *pwd-spy-sep-map* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-sep-map n $\equiv$
  case pwd-owner-iv n of (*s, S, A, B, C, D*) $\Rightarrow$
    (*insert* (*Spy*, *PubKey A*) *s, S, A, B, C, D*)

definition *pwd-spy-sep-agr* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-sep-agr n $\equiv$
  case pwd-spy-sep-map n of (*s, S, A, B, C, D*) $\Rightarrow$
    (*insert* (*Spy*, *PubKey C*) *s, S, A, B, C, D*)

definition *pwd-spy-sesk* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-sesk n $\equiv$
  let (*s, S, A, B, C, D*) = *pwd-spy-sep-agr n*;
    SK = (*Some S*, {*A, B*}, {*C, D*}) *in*
    (*insert* (*Spy*, *SesKey SK*) *s, S, A, B, C, D*)

definition *pwd-spy-mult* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-mult n $\equiv$
  case pwd-spy-sesk n of (*s, S, A, B, C, D*) $\Rightarrow$
    (*insert* (*Spy*, *Auth-PriK n* $\otimes$ *B*) *s, S, A, B, C, D*)

definition *pwd-spy-enc-pubk* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-enc-pubk n $\equiv$
  let (*s, S, A, B, C, D*) = *pwd-spy-mult n*; *SK* = (*Some S*, {*A, B*}, {*C, D*}) *in*
    (*insert* (*Spy*, *Crypt* (*SesK SK*) (*PubKey C*)) *s, S, A, B, C, D*)

definition *pwd-spy-enc-mult* :: *agent-id* ⇒ *stage* **where**
pwd-spy-enc-mult n ≡
  let (*s, S, A, B, C, D*) = *pwd-spy-enc-pubk n*;
    SK = (*Some S*, {*A, B*}, {*C, D*}) **in**
    (*insert* (*Spy, Crypt* (*SesK SK*) (*Auth-PriK n* ⊗ *B*)) *s, S, A, B, C, D*)

definition *pwd-spy-enc-sign* :: *agent-id* ⇒ *stage* **where**
pwd-spy-enc-sign n ≡
  let (*s, S, A, B, C, D*) = *pwd-spy-enc-mult n*;
    SK = (*Some S*, {*A, B*}, {*C, D*}) **in**
    (*insert* (*Spy, Crypt* (*SesK SK*) (*Sign n* (*Auth-PriK n*))) *s, S, A, B, C, D*)

definition *pwd-spy-con* :: *agent-id* ⇒ *stage* **where**
pwd-spy-con n ≡
  let (*s, S, A, B, C, D*) = *pwd-spy-enc-sign n*;
    SK = (*Some S*, {*A, B*}, {*C, D*}) **in**
    (*insert* (*Spy*, ⦃*Crypt* (*SesK SK*) (*Auth-PriK n* ⊗ *B*),
     Crypt (*SesK SK*) (*Sign n* (*Auth-PriK n*))⦄) *s, S, A, B, C, D*)

definition *pwd-spy-iv* :: *agent-id* ⇒ *stage* **where**
pwd-spy-iv n ≡
  let (*s, S, A, B, C, D*) = *pwd-spy-con n*; *SK* = (*Some S*, {*A, B*}, {*C, D*}) **in**
    (*insert* (*Spy, Token n* (*Auth-PriK n*) *B C SK*) *s, S, A, B, C, D*)

definition *pwd-owner-v* :: *agent-id* ⇒ *stage* **where**
pwd-owner-v n ≡ *stage-owner-v n* (*pwd-spy-iv n*)

definition *pwd-spy-dec* :: *agent-id* ⇒ *stage* **where**
pwd-spy-dec n ≡
  case *pwd-owner-v n* **of** (*s, S, A, B, C, D*) ⇒
   (*insert* (*Spy, Pwd n*) *s, S, A, B, C, D*)

definition *pwd-spy-id-prik* :: *agent-id* ⇒ *stage* **where**
pwd-spy-id-prik n ≡
  case *pwd-spy-dec n* **of** (*s, S, A, B, C, D*) ⇒
   (*insert* (*Spy*, ⟨*n, PriKey S*⟩) *s, S, A, B, C, D*)

definition *pwd-spy-id-pubk* :: *agent-id* ⇒ *stage* **where**
pwd-spy-id-pubk n ≡
  case *pwd-spy-id-prik n* **of** (*s, S, A, B, C, D*) ⇒
   (*insert* (*Spy*, ⟨*n, PubKey S*⟩) *s, S, A, B, C, D*)

definition *pwd-spy-id-sesk* :: *agent-id* ⇒ *stage* **where**
pwd-spy-id-sesk n ≡
  let (*s, S, A, B, C, D*) = *pwd-spy-id-pubk n*;
    SK = (*Some S*, {*A, B*}, {*C, D*}) **in**
    (*insert* (*Spy*, ⟨*n, SesKey SK*⟩) *s, S, A, B, C, D*)

definition *pwd-spy-id-pwd* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-id-pwd n $\equiv$
 case pwd-spy-id-sesk n of (*s, S, A, B, C, D*) $\Rightarrow$
  (*insert* (*Spy*, $\langle n, Pwd\ n\rangle$) *s, S, A, B, C, D*)

proposition *key-sets-crypts-subset*:
 $\llbracket U \in$ *key-sets X* (*crypts* (*Log* $-$ ' *spied H*)); $H \subseteq H'\rrbracket \implies$
  $U \in$ *key-sets X* (*crypts* (*Log* $-$ ' *spied H'*))
   (**is** $\llbracket$- $\in$ *?A*; -$\rrbracket \implies$ -)
$\langle proof \rangle$

fun *pwd-spy-i-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-i-state n (*S*, -) = {*Spy*} $\times$ ({*PriKey S, PubKey S, Key* (*Auth-ShaKey n*),
 Auth-PriKey n, Sign n (*Auth-PriK n*), *Crypt* (*Auth-ShaKey n*) (*PriKey S*),
 $\langle n$, *Key* (*Auth-ShaKey n*)$\rangle$} $\cup$ *range Num*)

proposition *pwd-spy-i-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\implies s_0 \models fst$ (*pwd-spy-i n*)
  (**is** - $\implies$ - $\models$ *?t*)
$\langle proof \rangle$

proposition *pwd-spy-i-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\implies$
  case pwd-spy-i n of (*s, S, A, B, C, D*) $\Rightarrow$
   pwd-spy-i-state n (*S, A, B, C, D*) $\subseteq s$
$\langle proof \rangle$

proposition *pwd-spy-i-unused*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\implies \exists A$. *PriKey A* $\notin$ *used* (*fst* (*pwd-spy-i n*))
$\langle proof \rangle$

fun *pwd-owner-ii-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-owner-ii-state n (*S, A, B, C, D*) =
 pwd-spy-i-state n (*S, A, B, C, D*) $\cup$ {*Owner n, Spy*} $\times$ {$\{\!|Num\ 1, PubKey\ A|\!\}$}

proposition *pwd-owner-ii-ex*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\implies$
  pred-owner-ii n (*pwd-spy-i n*) (*pwd-owner-ii n*)
$\langle proof \rangle$

proposition *pwd-owner-ii-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\implies s_0 \models fst$ (*pwd-owner-ii n*)
  (**is** - $\implies$ - $\models$ *?t*)
$\langle proof \rangle$

proposition *pwd-owner-ii-msg*:

$n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   case pwd-owner-ii n of $(s, S, A, B, C, D) \Rightarrow$
   pwd-owner-ii-state n $(S, A, B, C, D) \subseteq s \wedge$
   $\{Key\ (Auth\text{-}ShaKey\ n)\} \in$ *key-sets* $(PriKey\ S)\ (crypts\ (Log -`\ spied\ s))$
$\langle proof \rangle$

fun *pwd-spy-ii-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-ii-state n $(S, A, B, C, D) =$
  pwd-owner-ii-state n $(S, A, B, C, D) \cup \{Spy\} \times \{PriKey\ B,$
   $\{\!| Num\ 2, PubKey\ B |\!\}\}$

proposition *pwd-spy-ii-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* $(pwd\text{-}spy\text{-}ii\ n)$
  $(\textbf{is}\ \text{-} \Longrightarrow \text{-} \models \mathit{?t})$
$\langle proof \rangle$

proposition *pwd-spy-ii-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   case pwd-spy-ii n of $(s, S, A, B, C, D) \Rightarrow$
   pwd-spy-ii-state n $(S, A, B, C, D) \subseteq s \wedge$
   $\{Key\ (Auth\text{-}ShaKey\ n)\} \in$ *key-sets* $(PriKey\ S)\ (crypts\ (Log -`\ spied\ s))$
$\langle proof \rangle$

proposition *pwd-spy-ii-unused*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow \exists\, C.\ PriKey\ C \notin$ *used* $(fst\ (pwd\text{-}spy\text{-}ii\ n))$
$\langle proof \rangle$

fun *pwd-owner-iii-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-owner-iii-state n $(S, A, B, C, D) =$
  pwd-spy-ii-state n $(S, A, B, C, D) \cup \{Owner\ n, Spy\} \times \{\{\!| Num\ 3, PubKey\ C |\!\}\}$

proposition *pwd-owner-iii-ex*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   pred-owner-iii n $(pwd\text{-}spy\text{-}ii\ n)\ (pwd\text{-}owner\text{-}iii\ n)$
$\langle proof \rangle$

proposition *pwd-owner-iii-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* $(pwd\text{-}owner\text{-}iii\ n)$
  $(\textbf{is}\ \text{-} \Longrightarrow \text{-} \models \mathit{?t})$
$\langle proof \rangle$

proposition *pwd-owner-iii-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   case pwd-owner-iii n of $(s, S, A, B, C, D) \Rightarrow$
   pwd-owner-iii-state n $(S, A, B, C, D) \subseteq s \wedge$
   $\{Key\ (Auth\text{-}ShaKey\ n)\} \in$ *key-sets* $(PriKey\ S)\ (crypts\ (Log -`\ spied\ s))$
$\langle proof \rangle$

fun *pwd-spy-iii-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *state* **where**
pwd-spy-iii-state n (S, A, B, C, D) =
   pwd-owner-iii-state n (S, A, B, C, D) ∪ {*Spy*} × {*PriKey D*,
      {|*Num 4*, *PubKey D*|}}

proposition *pwd-spy-iii-rel*:
 n ∈ *bad-prikey* ∩ *bad-id-shakey* ⟹ s_0 ⊨ *fst (pwd-spy-iii n)*
   (**is** - ⟹ - ⊨ *?t*)
⟨*proof*⟩

proposition *pwd-spy-iii-msg*:
 n ∈ *bad-prikey* ∩ *bad-id-shakey* ⟹
   case pwd-spy-iii n of (s, S, A, B, C, D) ⇒
    pwd-spy-iii-state n (S, A, B, C, D) ⊆ *s* ∧
    {*Key (Auth-ShaKey n)*} ∈ *key-sets (PriKey S) (crypts (Log −' spied s))*
⟨*proof*⟩

fun *pwd-owner-iv-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *state* **where**
pwd-owner-iv-state n (S, A, B, C, D) = (*let SK = (Some S, {A, B}, {C, D}) in*
  insert (Owner n, SesKey SK) (pwd-spy-iii-state n (S, A, B, C, D))))

lemma *pwd-owner-iv-rel-1*:
 ⟦*n* ∈ *bad-prikey* ∩ *bad-id-shakey*; *pwd-spy-iii n = (s, S, A, B, C, D)*⟧ ⟹
   s_0 ⊨ *fst (pwd-owner-iv n)*
     (**is** ⟦-; -⟧ ⟹ - ⊨ *?t*)
⟨*proof*⟩

proposition *pwd-owner-iv-rel*:
 n ∈ *bad-prikey* ∩ *bad-id-shakey* ⟹ s_0 ⊨ *fst (pwd-owner-iv n)*
⟨*proof*⟩

proposition *pwd-owner-iv-msg*:
 n ∈ *bad-prikey* ∩ *bad-id-shakey* ⟹
   case pwd-owner-iv n of (s, S, A, B, C, D) ⇒
    pwd-owner-iv-state n (S, A, B, C, D) ⊆ *s* ∧
    {*Key (Auth-ShaKey n)*} ∈ *key-sets (PriKey S) (crypts (Log −' spied s))*
⟨*proof*⟩

fun *pwd-spy-sep-map-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *state* **where**
pwd-spy-sep-map-state n (S, A, B, C, D) =
   insert (Spy, PubKey A) (pwd-owner-iv-state n (S, A, B, C, D))

proposition *pwd-spy-sep-map-rel*:
 n ∈ *bad-prikey* ∩ *bad-id-shakey* ⟹ s_0 ⊨ *fst (pwd-spy-sep-map n)*
     (**is** - ⟹ - ⊨ *?t*)

⟨*proof*⟩

proposition *pwd-spy-sep-map-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   case pwd-spy-sep-map n of $(s, S, A, B, C, D) \Rightarrow$
    pwd-spy-sep-map-state n $(S, A, B, C, D) \subseteq s \wedge$
    $\{Key\ (Auth\text{-}ShaKey\ n)\} \in$ *key-sets* $(PriKey\ S)$ $(crypts\ (Log -\text{`}\ spied\ s))$
⟨*proof*⟩

fun *pwd-spy-sep-agr-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-sep-agr-state n $(S, A, B, C, D) =$
 insert $(Spy, PubKey\ C)$ $(pwd\text{-}spy\text{-}sep\text{-}map\text{-}state\ n\ (S, A, B, C, D))$

proposition *pwd-spy-sep-agr-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* $(pwd\text{-}spy\text{-}sep\text{-}agr\ n)$
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
⟨*proof*⟩

proposition *pwd-spy-sep-agr-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   case pwd-spy-sep-agr n of $(s, S, A, B, C, D) \Rightarrow$
    pwd-spy-sep-agr-state n $(S, A, B, C, D) \subseteq s \wedge$
    $\{Key\ (Auth\text{-}ShaKey\ n)\} \in$ *key-sets* $(PriKey\ S)$ $(crypts\ (Log -\text{`}\ spied\ s))$
⟨*proof*⟩

fun *pwd-spy-sesk-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-sesk-state n $(S, A, B, C, D) = ($*let* $SK = (Some\ S, \{A, B\}, \{C, D\})$ *in*
 insert $(Spy, SesKey\ SK)$ $(pwd\text{-}spy\text{-}sep\text{-}agr\text{-}state\ n\ (S, A, B, C, D)))$

proposition *pwd-spy-sesk-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* $(pwd\text{-}spy\text{-}sesk\ n)$
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
⟨*proof*⟩

proposition *pwd-spy-sesk-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   case pwd-spy-sesk n of $(s, S, A, B, C, D) \Rightarrow$
    pwd-spy-sesk-state n $(S, A, B, C, D) \subseteq s \wedge$
    $\{Key\ (Auth\text{-}ShaKey\ n)\} \in$ *key-sets* $(PriKey\ S)$ $(crypts\ (Log -\text{`}\ spied\ s))$
⟨*proof*⟩

fun *pwd-spy-mult-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-mult-state n $(S, A, B, C, D) =$
 insert $(Spy, Auth\text{-}PriK\ n \otimes B)$ $(pwd\text{-}spy\text{-}sesk\text{-}state\ n\ (S, A, B, C, D))$

proposition *pwd-spy-mult-rel*:

$n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* (*pwd-spy-mult n*)
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
$\langle$*proof*$\rangle$

proposition *pwd-spy-mult-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
  case pwd-spy-mult n of (*s, S, A, B, C, D*) $\Rightarrow$
   pwd-spy-mult-state n (*S, A, B, C, D*) $\subseteq s \wedge$
   {*Key* (*Auth-ShaKey n*)} $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* $-$' *spied s*))
$\langle$*proof*$\rangle$

fun *pwd-spy-enc-pubk-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-enc-pubk-state n (*S, A, B, C, D*) =
 (*let SK* = (*Some S*, {*A, B*}, {*C, D*}) *in*
 insert (*Spy, Crypt* (*SesK SK*) (*PubKey C*))
  (*pwd-spy-mult-state n* (*S, A, B, C, D*)))

proposition *pwd-spy-enc-pubk-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* (*pwd-spy-enc-pubk n*)
  (**is** - $\Longrightarrow$ - $\models$ *?t*)
$\langle$*proof*$\rangle$

proposition *pwd-spy-enc-pubk-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
  case pwd-spy-enc-pubk n of (*s, S, A, B, C, D*) $\Rightarrow$
   pwd-spy-enc-pubk-state n (*S, A, B, C, D*) $\subseteq s \wedge$
   {*Key* (*Auth-ShaKey n*)} $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* $-$' *spied s*))
$\langle$*proof*$\rangle$

fun *pwd-spy-enc-mult-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-enc-mult-state n (*S, A, B, C, D*) =
 (*let SK* = (*Some S*, {*A, B*}, {*C, D*}) *in*
 insert (*Spy, Crypt* (*SesK SK*) (*Auth-PriK n* $\otimes$ *B*))
  (*pwd-spy-enc-pubk-state n* (*S, A, B, C, D*)))

proposition *pwd-spy-enc-mult-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* (*pwd-spy-enc-mult n*)
  (**is** - $\Longrightarrow$ - $\models$ *?t*)
$\langle$*proof*$\rangle$

proposition *pwd-spy-enc-mult-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
  case pwd-spy-enc-mult n of (*s, S, A, B, C, D*) $\Rightarrow$
   pwd-spy-enc-mult-state n (*S, A, B, C, D*) $\subseteq s \wedge$
   {*Key* (*Auth-ShaKey n*)} $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* $-$' *spied s*))
$\langle$*proof*$\rangle$

fun *pwd-spy-enc-sign-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-enc-sign-state n $(S, A, B, C, D) =$
  (*let* $SK = (Some\ S, \{A, B\}, \{C, D\})$ *in*
  insert $(Spy,\ Crypt\ (SesK\ SK)\ (Sign\ n\ (Auth\text{-}PriK\ n)))$
    $(pwd\text{-}spy\text{-}enc\text{-}mult\text{-}state\ n\ (S, A, B, C, D)))$

proposition *pwd-spy-enc-sign-rel*:
  $n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}enc\text{-}sign\ n)$
    (**is** - $\Longrightarrow$ - $\models$ *?t*)
$\langle proof \rangle$

proposition *pwd-spy-enc-sign-msg*:
  $n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
    case pwd-spy-enc-sign n *of* $(s, S, A, B, C, D) \Rightarrow$
      pwd-spy-enc-sign-state n $(S, A, B, C, D) \subseteq s\ \wedge$
      $\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -\ `\ spied\ s))$
$\langle proof \rangle$

fun *pwd-spy-con-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-con-state n $(S, A, B, C, D) = (let\ SK = (Some\ S, \{A, B\}, \{C, D\})$ *in*
  insert $(Spy, \{\!|Crypt\ (SesK\ SK)\ (Auth\text{-}PriK\ n \otimes B),$
    $Crypt\ (SesK\ SK)\ (Sign\ n\ (Auth\text{-}PriK\ n))|\!\})$
    $(pwd\text{-}spy\text{-}enc\text{-}sign\text{-}state\ n\ (S, A, B, C, D)))$

proposition *pwd-spy-con-rel*:
  $n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}con\ n)$
    (**is** - $\Longrightarrow$ - $\models$ *?t*)
$\langle proof \rangle$

proposition *pwd-spy-con-msg*:
  $n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
    case pwd-spy-con n *of* $(s, S, A, B, C, D) \Rightarrow$
      pwd-spy-con-state n $(S, A, B, C, D) \subseteq s\ \wedge$
      $\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -\ `\ spied\ s))$
$\langle proof \rangle$

fun *pwd-spy-iv-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-iv-state n $(S, A, B, C, D) = (let\ SK = (Some\ S, \{A, B\}, \{C, D\})$ *in*
  insert $(Spy,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK)$
    $(pwd\text{-}spy\text{-}con\text{-}state\ n\ (S, A, B, C, D)))$

proposition *pwd-spy-iv-rel*:
  $n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}iv\ n)$
    (**is** - $\Longrightarrow$ - $\models$ *?t*)
$\langle proof \rangle$

proposition *pwd-spy-iv-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   case pwd-spy-iv n of (*s*, *S*, *A*, *B*, *C*, *D*) $\Rightarrow$
   pwd-spy-iv-state n (*S*, *A*, *B*, *C*, *D*) $\subseteq$ *s* $\land$
   {*Key* (*Auth-ShaKey n*)} $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* $-$' *spied s*))
⟨*proof*⟩

fun *pwd-owner-v-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-owner-v-state n (*S*, *A*, *B*, *C*, *D*) = (*let SK* = (*Some S*, {*A*, *B*}, {*C*, *D*}) *in*
  insert (*Spy*, *Crypt* (*SesK SK*) (*Pwd n*)) (*pwd-spy-iv-state n* (*S*, *A*, *B*, *C*, *D*)))

proposition *pwd-owner-v-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$ $s_0 \models$ *fst* (*pwd-owner-v n*)
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
⟨*proof*⟩

proposition *pwd-owner-v-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   let (*s*, *S*, *A*, *B*, *C*, *D*) = *pwd-owner-v n*; *SK* = (*Some S*, {*A*, *B*}, {*C*, *D*}) *in*
   pwd-owner-v-state n (*S*, *A*, *B*, *C*, *D*) $\subseteq$ *s* $\land$
   {*Key* (*Auth-ShaKey n*)} $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* $-$' *spied s*)) $\land$
   {*SesKey SK*} $\in$ *key-sets* (*Pwd n*) (*crypts* (*Log* $-$' *spied s*))
⟨*proof*⟩

abbreviation *pwd-spy-dec-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-dec-state n x $\equiv$ *insert* (*Spy*, *Pwd n*) (*pwd-owner-v-state n x*)

proposition *pwd-spy-dec-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$ $s_0 \models$ *fst* (*pwd-spy-dec n*)
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
⟨*proof*⟩

proposition *pwd-spy-dec-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   let (*s*, *S*, *A*, *B*, *C*, *D*) = *pwd-spy-dec n*; *SK* = (*Some S*, {*A*, *B*}, {*C*, *D*}) *in*
   pwd-spy-dec-state n (*S*, *A*, *B*, *C*, *D*) $\subseteq$ *s* $\land$
   {*Key* (*Auth-ShaKey n*)} $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* $-$' *spied s*)) $\land$
   {*SesKey SK*} $\in$ *key-sets* (*Pwd n*) (*crypts* (*Log* $-$' *spied s*))
⟨*proof*⟩

fun *pwd-spy-id-prik-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-id-prik-state n (*S*, *A*, *B*, *C*, *D*) =
  insert (*Spy*, ⟨*n*, *PriKey S*⟩) (*pwd-spy-dec-state n* (*S*, *A*, *B*, *C*, *D*))

proposition *pwd-spy-id-prik-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$ $s_0 \models$ *fst* (*pwd-spy-id-prik n*)

(**is** - $\Longrightarrow$ - $\models$ *?t*)
⟨*proof*⟩

proposition *pwd-spy-id-prik-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   let $(s, S, A, B, C, D) = $ *pwd-spy-id-prik* n;
    $SK = (Some\ S,\ \{A,\ B\},\ \{C,\ D\})$ *in*
    pwd-spy-id-prik-state $n\ (S,\ A,\ B,\ C,\ D) \subseteq s\ \wedge$
    $\{SesKey\ SK\} \in$ *key-sets* $(Pwd\ n)\ (crypts\ (Log\ -'\ spied\ s))$
⟨*proof*⟩

fun *pwd-spy-id-pubk-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-id-pubk-state $n\ (S,\ A,\ B,\ C,\ D) = $
 insert $(Spy,\ \langle n,\ PubKey\ S\rangle)\ (pwd\text{-}spy\text{-}id\text{-}prik\text{-}state\ n\ (S,\ A,\ B,\ C,\ D))$

proposition *pwd-spy-id-pubk-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* $(pwd\text{-}spy\text{-}id\text{-}pubk\ n)$
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
⟨*proof*⟩

proposition *pwd-spy-id-pubk-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   let $(s, S, A, B, C, D) = $ *pwd-spy-id-pubk* n;
    $SK = (Some\ S,\ \{A,\ B\},\ \{C,\ D\})$ *in*
    pwd-spy-id-pubk-state $n\ (S,\ A,\ B,\ C,\ D) \subseteq s\ \wedge$
    $\{SesKey\ SK\} \in$ *key-sets* $(Pwd\ n)\ (crypts\ (Log\ -'\ spied\ s))$
⟨*proof*⟩

fun *pwd-spy-id-sesk-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**
pwd-spy-id-sesk-state $n\ (S,\ A,\ B,\ C,\ D) = $
 (*let* $SK = (Some\ S,\ \{A,\ B\},\ \{C,\ D\})$ *in*
 insert $(Spy,\ \langle n,\ SesKey\ SK\rangle)\ (pwd\text{-}spy\text{-}id\text{-}pubk\text{-}state\ n\ (S,\ A,\ B,\ C,\ D)))$

proposition *pwd-spy-id-sesk-rel*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* $(pwd\text{-}spy\text{-}id\text{-}sesk\ n)$
   (**is** - $\Longrightarrow$ - $\models$ *?t*)
⟨*proof*⟩

proposition *pwd-spy-id-sesk-msg*:
 $n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow$
   let $(s, S, A, B, C, D) = $ *pwd-spy-id-sesk* n;
    $SK = (Some\ S,\ \{A,\ B\},\ \{C,\ D\})$ *in*
    pwd-spy-id-sesk-state $n\ (S,\ A,\ B,\ C,\ D) \subseteq s\ \wedge$
    $\{SesKey\ SK\} \in$ *key-sets* $(Pwd\ n)\ (crypts\ (Log\ -'\ spied\ s))$
⟨*proof*⟩

abbreviation *pwd-spy-id-pwd-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *state* **where**
pwd-spy-id-pwd-state n x ≡ *insert* (*Spy*, ⟨*n*, *Pwd n*⟩) (*pwd-spy-id-sesk-state n x*)

proposition *pwd-spy-id-pwd-rel*:
  n ∈ *bad-prikey* ∩ *bad-id-shakey* ⟹ *s₀* ⊨ *fst* (*pwd-spy-id-pwd n*)
    (**is** - ⟹ - ⊨ *?t*)
⟨*proof*⟩

proposition *pwd-spy-id-pwd-msg*:
  n ∈ *bad-prikey* ∩ *bad-id-shakey* ⟹
    case pwd-spy-id-pwd n of (*s*, *S*, *A*, *B*, *C*, *D*) ⇒
      pwd-spy-id-pwd-state n (*S*, *A*, *B*, *C*, *D*) ⊆ *s*
⟨*proof*⟩

theorem *pwd-compromised*:
  n ∈ *bad-prikey* ∩ *bad-id-shakey* ⟹ ∃ *s*. *s₀* ⊨ *s* ∧ {*Pwd n*, ⟨*n*, *Pwd n*⟩} ⊆ *spied s*
⟨*proof*⟩

end

References

[1] International Civil Aviation Organization (ICAO). *Doc 9303 – Machine Readable Travel Documents – Part 11: Security Mechanisms for MRTDs*, 7th edition, 2015.

[2] A. Krauss. *Defining Recursive Functions in Isabelle/HOL*. https://isabelle.in.tum.de/website-Isabelle2020/dist/Isabelle2020/doc/functions.pdf.

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

[4] T. Nipkow. *Programming and Proving in Isabelle/HOL*, Apr. 2020. https://isabelle.in.tum.de/website-Isabelle2020/dist/Isabelle2020/doc/prog-prove.pdf.

[5] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, Apr. 2020. https://isabelle.in.tum.de/website-Isabelle2020/dist/Isabelle2020/doc/tutorial.pdf.

[6] P. Noce. Verification of a diffie-hellman password-based authentication protocol by extending the inductive method. *Archive of Formal Proofs*, Jan. 2017. http://isa-afp.org/entries/Password_Authentication_Protocol.html, Formal proof development.

[7] L. C. Paulson. The inductive approach to verifying cryptographic pro-
    tocols. *Journal of Computer Security*, Dec. 1998.