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

February 23, 2021

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

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 $\mathit{Nonce}\ n \notin H \Longrightarrow \mathit{Nonce}\ n \in \mathit{synth}\ H$ or $\mathit{PriKey}\ A \notin H \Longrightarrow \mathit{PriKey}\ A \in \mathit{synth}\ H$. 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*:

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

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 $[\![s \models s';\ P\ s'[;\ Q]]\!] \Longrightarrow R\ s'$ 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 $s = s_0$, $P = (\lambda s.\ X \in \mathit{parts}\ (\mathit{used}\ s))$ 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 $[\![s \vdash s';\ P\ s';\ \neg\ P\ s[;\ Q]]\!] \Longrightarrow R\ s'$ 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 $\langle n,\ X_n\rangle \notin \mathit{spied}\ s$, 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 $(\mathit{Spy},\ \langle n,\ C\ X_1\ \ldots\ X_m\rangle)$, with event $(\mathit{Spy},\ \langle n,\ X_i\rangle)$, where $1 \leq i \leq m$ and C is some constructor of datatype *msg*, or conversely augments a state containing event $(\mathit{Spy},\ \langle n,\ X_i\rangle)$ with $(\mathit{Spy},\ \langle n,\ C\ X_1\ \ldots\ X_m\rangle)$. 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 $H \subseteq H'$ 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 $U \in$ *key-sets* X (*crypts* (*Log* −‘ *spied* s)) and $U \subseteq$ *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. $\mathit{Asset}\ n \rightarrow \mathit{Owner}\ n$:
   $\mathit{Crypt}\ (\mathit{Auth\text{-}ShaKey}\ n)\ (\mathit{PriKey}\ S)$

2. $\mathit{Owner}\ n \rightarrow \mathit{Asset}\ n$:
   $\{\!|\mathit{Num}\ \mathit{1},\ \mathit{PubKey}\ A|\!\}$

3. $\mathit{Asset}\ n \rightarrow \mathit{Owner}\ n$:
   $\{\!|\mathit{Num}\ \mathit{2},\ \mathit{PubKey}\ B|\!\}$

4. $\mathit{Owner}\ n \rightarrow \mathit{Asset}\ n$:
   $\{\!|\mathit{Num}\ \mathit{3},\ \mathit{PubKey}\ C|\!\}$

5. $\mathit{Asset}\ n \rightarrow \mathit{Owner}\ n$:
   $\{\!|\mathit{Num}\ \mathit{4},\ \mathit{PubKey}\ D|\!\}$

6. $\mathit{Owner}\ n \rightarrow \mathit{Asset}\ n$:
   $\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{PubKey}\ D)$

7. $\mathit{Asset}\ n \rightarrow \mathit{Owner}\ n$:
   $\{\!|\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{PubKey}\ C),$
   $\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Auth\text{-}PriK}\ n \otimes B),$
   $\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Crypt}\ \mathit{SigK}$
   $\{\!|\mathit{Hash}\ (\mathit{Agent}\ n),\ \mathit{Hash}\ (\mathit{Auth\text{-}PubKey}\ n)|\!\})|\!\}$

8. $\mathit{Owner}\ n \rightarrow \mathit{Asset}\ n$:
   $\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Pwd}\ n)$

9. $\mathit{Asset}\ n \rightarrow \mathit{Owner}\ n$:
   $\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Num}\ \mathit{0})$

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* $\otimes$ *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* $\neq$ *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 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 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 $\langle n, \mathit{Agent}\ n\rangle$ 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 $\langle n, \mathit{Crypt}\ K\ X\rangle$ from $\langle n, X\rangle$ or $\langle n, \mathit{Key}\ K\rangle$ and vice versa. A single protocol rule is rather defined, which enables the spy to infer $\langle n, X\rangle$ from $\langle n, \mathit{Key}\ K\rangle$ 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 $\mathit{Key}\ (\mathit{SesK}\ \mathit{SK})$. 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 $\langle n, \mathit{Key}\ (\mathit{SesK}\ \mathit{SK})\rangle$, viz. she need not know $\langle n, \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ X_1\rangle$ 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.

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.

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.

- 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** $\otimes$ *70*) |
  Hash msg |
  Crypt key msg |
  MPair msg msg |
  IDInfo agent-id msg |
  Log msg

syntax
  $\textit{-MPair} :: [\textit{'a}, \textit{args}] \Rightarrow \textit{'a} * \textit{'b} \quad ((\textit{2}\{\!|\text{-,/ -}|\!\}))$
  $\textit{-IDInfo} :: [\textit{agent-id}, \textit{msg}] \Rightarrow \textit{msg} \qquad ((\textit{2}\langle\text{-,/ -}\rangle))$
translations
  $\{\!|X,\ Y,\ Z|\!\} \rightleftharpoons \{\!|X,\ \{\!|Y,\ Z|\!\}|\!\}$
  $\{\!|X,\ Y|\!\} \rightleftharpoons \textit{CONST MPair X Y}$
  $\langle n,\ X\rangle \rightleftharpoons \textit{CONST IDInfo n X}$

abbreviation *SigKey* :: *msg* **where**
SigKey $\equiv$ *Key SigK*

abbreviation *VerKey* :: *msg* **where**
VerKey $\equiv$ *Key VerK*

abbreviation *PriKey* :: *key-id* $\Rightarrow$ *msg* **where**
PriKey $\equiv$ *Key* $\circ$ *PriK*

abbreviation *PubKey* :: *key-id* $\Rightarrow$ *msg* **where**
PubKey $\equiv$ *Key* $\circ$ *PubK*

abbreviation *ShaKey* :: *key-id* $\Rightarrow$ *msg* **where**
ShaKey $\equiv$ *Key* $\circ$ *ShaK*

abbreviation *SesKey* :: *seskey-in* $\Rightarrow$ *msg* **where**
SesKey $\equiv$ *Key* $\circ$ *SesK*

primrec *InvK* :: *key* $\Rightarrow$ *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* $\Rightarrow$ *msg* **where**

$InvKey \equiv Key \circ InvK$

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

parts-used [*intro*]:
$X \in H \Longrightarrow X \in parts\ H \mid$

parts-crypt [*intro*]:
$Crypt\ K\ X \in parts\ H \Longrightarrow X \in parts\ H \mid$

parts-fst [*intro*]:
$\{\!|X,\ Y|\!\} \in parts\ H \Longrightarrow X \in parts\ H \mid$

parts-snd [*intro*]:
$\{\!|X,\ Y|\!\} \in parts\ H \Longrightarrow Y \in parts\ H$

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

crypts-used [*intro*]:
$X \in H \Longrightarrow X \in crypts\ H \mid$

crypts-hash [*intro*]:
$foldr\ Crypt\ KS\ (Hash\ X) \in crypts\ H \Longrightarrow foldr\ Crypt\ KS\ X \in crypts\ H \mid$

crypts-fst [*intro*]:
$foldr\ Crypt\ KS\ \{\!|X,\ Y|\!\} \in crypts\ H \Longrightarrow foldr\ Crypt\ KS\ X \in crypts\ H \mid$

crypts-snd [*intro*]:
$foldr\ Crypt\ KS\ \{\!|X,\ Y|\!\} \in crypts\ H \Longrightarrow foldr\ Crypt\ KS\ Y \in crypts\ H$

definition *key-sets* :: *msg* ⇒ *msg set* ⇒ *msg set set* **where**
$key\text{-}sets\ X\ H \equiv \{InvKey\ `\ set\ KS \mid KS.\ foldr\ Crypt\ KS\ X \in H\}$

definition *parts-msg* :: *msg* ⇒ *msg set* **where**
$parts\text{-}msg\ X \equiv parts\ \{X\}$

definition *crypts-msg* :: *msg* ⇒ *msg set* **where**
$crypts\text{-}msg\ X \equiv crypts\ \{X\}$

definition *key-sets-msg* :: *msg* ⇒ *msg* ⇒ *msg set set* **where**
$key\text{-}sets\text{-}msg\ X\ Y \equiv key\text{-}sets\ X\ \{Y\}$

fun *seskey-set* :: *seskey-in* ⇒ *key-id set* **where**
$seskey\text{-}set\ (Some\ S,\ U,\ V) = insert\ S\ (U \cup V) \mid$

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 $\equiv$ *bad-id-pwd* $\cap$ *bad-pwd*

abbreviation *bad-id-prikey* :: *agent-id set* **where**
bad-id-prikey $\equiv$ (*bad-agent* $\cup$ *bad-id-pubk* $\cup$ *bad-id-prik*) $\cap$ *bad-prikey*

abbreviation *bad-id-pubkey* :: *agent-id set* **where**
bad-id-pubkey $\equiv$ *bad-agent* $\cup$ *bad-id-pubk* $\cup$ *bad-id-prik* $\cap$ *bad-prikey*

abbreviation *bad-id-shakey* :: *agent-id set* **where**
bad-id-shakey $\equiv$ *bad-id-shak* $\cap$ *bad-shakey*

type-synonym *event* = *agent* $\times$ *msg*

type-synonym *state* = *event set*

abbreviation *used* :: *state* $\Rightarrow$ *msg set* **where**
used s $\equiv$ *Range s*

abbreviation *spied* :: *state* $\Rightarrow$ *msg set* **where**
spied s $\equiv$ *s* `` {*Spy*}

abbreviation s_0 :: *state* **where**
s_0 $\equiv$ *range* (λ*n*. (*Asset n*, *Auth-PriKey n*)) $\cup$ {*Spy*} $\times$ *insert VerKey*
(*range Num* $\cup$ *range Auth-PubKey* $\cup$ *range* (λ*n*. *Sign n* (*Auth-PriK n*)) $\cup$
Agent ` *bad-agent* $\cup$ *Pwd* ` *bad-pwd* $\cup$ *PriKey* ` *bad-prik* $\cup$ *ShaKey* ` *bad-shak* $\cup$
(λ*n*. $\langle$*n*, *Pwd n*$\rangle$) ` *bad-id-password* $\cup$
(λ*n*. $\langle$*n*, *Auth-PriKey n*$\rangle$) ` *bad-id-prikey* $\cup$
(λ*n*. $\langle$*n*, *Auth-PubKey n*$\rangle$) ` *bad-id-pubkey* $\cup$
(λ*n*. $\langle$*n*, *Key* (*Auth-ShaKey n*)$\rangle$) ` *bad-id-shakey*)

abbreviation *rel-asset-i* :: (*state* $\times$ *state*) *set* **where**
rel-asset-i $\equiv$ {(*s*, *s'*) | *s s' n S*.
s' = *insert* (*Asset n*, *PriKey S*) *s* $\cup$
{*Asset n*, *Spy*} $\times$ {*Crypt* (*Auth-ShaKey n*) (*PriKey S*)} $\cup$
{(*Spy*, *Log* (*Crypt* (*Auth-ShaKey n*) (*PriKey S*)))} $\wedge$
PriKey S $\notin$ *used s*}

abbreviation *rel-owner-ii* :: (*state* $\times$ *state*) *set* **where**
rel-owner-ii $\equiv$ {(*s*, *s'*) | *s s' n S A K*.
s' = *insert* (*Owner n*, *PriKey A*) *s* $\cup$
{*Owner n*, *Spy*} $\times$ {$\{\!|$*Num 1*, *PubKey A*$|\!\}$} $\cup$
{*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* $\times$ *state*) *set* **where**
rel-asset-ii $\equiv$ {(*s*, *s'*) | *s s' n A B*.

$s' = insert\ (Asset\ n,\ PriKey\ B)\ s \cup$
$\{Asset\ n,\ Spy\} \times \{\{\!|Num\ 2,\ PubKey\ B|\!\}\} \cup$
$\{Spy\} \times Log\ `\ \{\{\!|Num\ 1,\ PubKey\ A|\!\},\ \{\!|Num\ 2,\ PubKey\ B|\!\}\} \wedge$
$\{\!|Num\ 1,\ PubKey\ A|\!\} \in used\ s \wedge$
$PriKey\ B \notin used\ s\}$

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

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

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

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

abbreviation *rel-owner-v* :: (*state* × *state*) *set* **where**
$rel\text{-}owner\text{-}v \equiv \{(s,\ s') \mid s\ s'\ n\ A\ B\ C\ SK.$
$s' = s \cup \{Owner\ n,\ Spy\} \times \{Crypt\ (SesK\ SK)\ (Pwd\ n)\} \cup$
$\{Spy\} \times Log\ `\ \{Token\ n\ A\ B\ C\ SK,\ Crypt\ (SesK\ SK)\ (Pwd\ n)\} \wedge$
$Token\ n\ A\ B\ C\ SK \in used\ s \wedge$

$(Owner\ n,\ SesKey\ SK) \in s \wedge$
$B \in fst\ (snd\ SK)\}$

abbreviation *rel-asset-v* :: (*state* × *state*) *set* **where**
$rel\text{-}asset\text{-}v \equiv \{(s,\ s') \mid s\ s'\ n\ SK.$
$s' = s \cup \{Asset\ n,\ Spy\} \times \{Crypt\ (SesK\ SK)\ (Num\ 0)\} \cup$
$\{Spy\} \times Log\ `\ \{Crypt\ (SesK\ SK)\ (Pwd\ n),\ Crypt\ (SesK\ SK)\ (Num\ 0)\} \wedge$
$(Asset\ n,\ SesKey\ SK) \in s \wedge$
$Crypt\ (SesK\ SK)\ (Pwd\ n) \in used\ s\}$

abbreviation *rel-prik* :: (*state* × *state*) *set* **where**
$rel\text{-}prik \equiv \{(s,\ s') \mid s\ s'\ A.$
$s' = insert\ (Spy,\ PriKey\ A)\ s \wedge$
$PriKey\ A \notin used\ s\}$

abbreviation *rel-pubk* :: (*state* × *state*) *set* **where**
$rel\text{-}pubk \equiv \{(s,\ s') \mid s\ s'\ A.$
$s' = insert\ (Spy,\ PubKey\ A)\ s \wedge$
$PriKey\ A \in spied\ s\}$

abbreviation *rel-sesk* :: (*state* × *state*) *set* **where**
$rel\text{-}sesk \equiv \{(s,\ s') \mid s\ s'\ A\ B\ C\ D\ S.$
$s' = insert\ (Spy,\ SesKey\ (Some\ S,\ \{A,\ B\},\ \{C,\ D\}))\ s \wedge$
$\{PriKey\ S,\ PriKey\ A,\ PubKey\ B,\ PriKey\ C,\ PubKey\ D\} \subseteq spied\ s\}$

abbreviation *rel-fact* :: (*state* × *state*) *set* **where**
$rel\text{-}fact \equiv \{(s,\ s') \mid s\ s'\ A\ B.$
$s' = s \cup \{Spy\} \times \{PriKey\ A,\ PriKey\ B\} \wedge$
$A \otimes B \in spied\ s \wedge$
$(PriKey\ A \in spied\ s \vee PriKey\ B \in spied\ s)\}$

abbreviation *rel-mult* :: (*state* × *state*) *set* **where**
$rel\text{-}mult \equiv \{(s,\ s') \mid s\ s'\ A\ B.$
$s' = insert\ (Spy,\ A \otimes B)\ s \wedge$
$\{PriKey\ A,\ PriKey\ B\} \subseteq spied\ s\}$

abbreviation *rel-hash* :: (*state* × *state*) *set* **where**
$rel\text{-}hash \equiv \{(s,\ s') \mid s\ s'\ X.$
$s' = insert\ (Spy,\ Hash\ X)\ s \wedge$
$X \in spied\ s\}$

abbreviation *rel-dec* :: (*state* × *state*) *set* **where**
$rel\text{-}dec \equiv \{(s,\ s') \mid s\ s'\ K\ X.$
$s' = insert\ (Spy,\ X)\ s \wedge$
$\{Crypt\ K\ X,\ InvKey\ K\} \subseteq spied\ s\}$

abbreviation *rel-enc* :: (*state* × *state*) *set* **where**
$rel\text{-}enc \equiv \{(s,\ s') \mid s\ s'\ K\ X.$

$s' = insert\ (Spy,\ Crypt\ K\ X)\ s\ \wedge$
$\{X,\ Key\ K\} \subseteq spied\ s\}$

abbreviation *rel-sep* :: (*state* $\times$ *state*) *set* **where**
$rel\text{-}sep \equiv \{(s,\ s') \mid s\ s'\ X\ Y.$
$s' = s \cup \{Spy\} \times \{X,\ Y\} \wedge$
$\{\!|X,\ Y|\!\} \in spied\ s\}$

abbreviation *rel-con* :: (*state* $\times$ *state*) *set* **where**
$rel\text{-}con \equiv \{(s,\ s') \mid s\ s'\ X\ Y.$
$s' = insert\ (Spy,\ \{\!|X,\ Y|\!\})\ s\ \wedge$
$\{X,\ Y\} \subseteq spied\ s\}$

abbreviation *rel-id-agent* :: (*state* $\times$ *state*) *set* **where**
$rel\text{-}id\text{-}agent \equiv \{(s,\ s') \mid s\ s'\ n.$
$s' = insert\ (Spy,\ \langle n,\ Agent\ n\rangle)\ s\ \wedge$
$Agent\ n \in spied\ s\}$

abbreviation *rel-id-invk* :: (*state* $\times$ *state*) *set* **where**
$rel\text{-}id\text{-}invk \equiv \{(s,\ s') \mid s\ s'\ n\ K.$
$s' = insert\ (Spy,\ \langle n,\ InvKey\ K\rangle)\ s\ \wedge$
$\{InvKey\ K,\ \langle n,\ Key\ K\rangle\} \subseteq spied\ s\}$

abbreviation *rel-id-sesk* :: (*state* $\times$ *state*) *set* **where**
$rel\text{-}id\text{-}sesk \equiv \{(s,\ s') \mid s\ s'\ n\ A\ SK\ X\ U.$
$s' = s \cup \{Spy\} \times \{\langle n,\ PubKey\ A\rangle,\ \langle n,\ SesKey\ SK\rangle\} \wedge$
$\{PubKey\ A,\ SesKey\ SK\} \subseteq spied\ s\ \wedge$
$(\langle n,\ PubKey\ A\rangle \in spied\ s \vee \langle n,\ SesKey\ SK\rangle \in spied\ s)\ \wedge$
$A \in seskey\text{-}set\ SK\ \wedge$
$SesKey\ SK \in U\ \wedge$
$U \in key\text{-}sets\ X\ (crypts\ (Log\ -`\ spied\ s))\}$

abbreviation *rel-id-fact* :: (*state* $\times$ *state*) *set* **where**
$rel\text{-}id\text{-}fact \equiv \{(s,\ s') \mid s\ s'\ n\ A\ B.$
$s' = s \cup \{Spy\} \times \{\langle n,\ PriKey\ A\rangle,\ \langle n,\ PriKey\ B\rangle\} \wedge$
$\{PriKey\ A,\ PriKey\ B,\ \langle n,\ A \otimes B\rangle\} \subseteq spied\ s\}$

abbreviation *rel-id-mult* :: (*state* $\times$ *state*) *set* **where**
$rel\text{-}id\text{-}mult \equiv \{(s,\ s') \mid s\ s'\ n\ A\ B\ U.$
$s' = insert\ (Spy,\ \langle n,\ A \otimes B\rangle)\ s\ \wedge$
$U \cup \{PriKey\ A,\ PriKey\ B,\ A \otimes B\} \subseteq spied\ s\ \wedge$
$(\langle n,\ PriKey\ A\rangle \in spied\ s \vee \langle n,\ PriKey\ B\rangle \in spied\ s)\ \wedge$
$U \in key\text{-}sets\ (A \otimes B)\ (crypts\ (Log\ -`\ spied\ s))\}$

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

$(\langle n,\ X\rangle \in \mathit{spied}\ s \lor \langle n,\ \mathit{Hash}\ X\rangle \in \mathit{spied}\ s) \land$
$U \in \mathit{key\text{-}sets}\ (\mathit{Hash}\ X)\ (\mathit{crypts}\ (\mathit{Log}\ -\text{‘}\ \mathit{spied}\ s))\}$

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

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

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

definition *rel* :: (*state* × *state*) *set* **where**
rel ≡ *rel-asset-i* ∪ *rel-owner-ii* ∪ *rel-asset-ii* ∪ *rel-owner-iii* ∪
rel-asset-iii ∪ *rel-owner-iv* ∪ *rel-asset-iv* ∪ *rel-owner-v* ∪ *rel-asset-v* ∪
rel-prik ∪ *rel-pubk* ∪ *rel-sesk* ∪ *rel-fact* ∪ *rel-mult* ∪ *rel-hash* ∪ *rel-dec* ∪
rel-enc ∪ *rel-sep* ∪ *rel-con* ∪ *rel-id-agent* ∪ *rel-id-invk* ∪ *rel-id-sesk* ∪
rel-id-fact ∪ *rel-id-mult* ∪ *rel-id-hash* ∪ *rel-id-crypt* ∪ *rel-id-sep* ∪ *rel-id-con*

abbreviation *in-rel* :: *state* ⇒ *state* ⇒ *bool* (**infix** ⊢ *60*) **where**
$s \vdash s' \equiv (s,\ s') \in \mathit{rel}$

abbreviation *in-rel-rtrancl* :: *state* ⇒ *state* ⇒ *bool* (**infix** ⊨ *60*) **where**
$s \models s' \equiv (s,\ s') \in \mathit{rel}^*$

end

2 Confidentiality and authenticity properties

theory *Authentication*
imports *Definitions*
begin

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

proof (*erule rtrancl-induct*, *simp*, (*rule impI*)+)
  fix *y z*
  assume
    A: $(x, y) \in r^*$ **and**
    B: $(y, z) \in r$ **and**
    C: *P z*
  assume $P\ y \longrightarrow \neg\ P\ x \longrightarrow (\exists\, u\ v.\ ?Q\ x\ y\ u\ v)$ **and** $\neg\ P\ x$
  hence *D*: $P\ y \longrightarrow (\exists\, u\ v.\ ?Q\ x\ y\ u\ v)$ **by** *simp*
  show $\exists\, u\ v.\ ?Q\ x\ z\ u\ v$
  proof (*cases P y*)
    case *True*
    with *D* **obtain** *u v* **where** *?Q x y u v* **by** *blast*
    moreover from *this* **and** *B* **have** $(v, z) \in r^*$ **by** *auto*
    ultimately show *?thesis* **by** *blast*
  next
    case *False*
    with *A* **and** *B* **and** *C* **have** *?Q x z y z* **by** *simp*
    thus *?thesis* **by** *blast*
  qed
qed

proposition *state-subset*:
 $s \models s' \Longrightarrow s \subseteq s'$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def image-def*)

proposition *spied-subset*:
 $s \models s' \Longrightarrow spied\ s \subseteq spied\ s'$
by (*rule Image-mono*, *erule state-subset*, *simp*)

proposition *used-subset*:
 $s \models s' \Longrightarrow used\ s \subseteq used\ s'$
by (*rule Range-mono*, *rule state-subset*)

proposition *asset-ii-init*:
 $[\![s_0 \models s;\ (Asset\ n,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s]\!] \Longrightarrow$
    $PriKey\ A \notin spied\ s_0$
by (*drule rtrancl-start*, *assumption*, *simp add*: *image-def*, (*erule exE*)+,
 erule conjE, *rule notI*, *drule spied-subset*, *drule subsetD*, *assumption*,
 auto simp add: *rel-def*)

proposition *auth-prikey-used*:
 $s_0 \models s \Longrightarrow Auth\text{-}PriKey\ n \in used\ s$
by (*drule used-subset*, *erule subsetD*, *simp add*: *Range-iff image-def*, *blast*)

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

by (*erule rtrancl-induct*, *auto simp add*: *rel-def image-def*)

proposition *owner-ii-used*:
$s_0 \models s \Longrightarrow$
$(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \longrightarrow$
$PriKey\ A \in used\ s$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def image-def*)

proposition *asset-ii-used*:
$s_0 \models s \Longrightarrow$
$(Asset\ n,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
$PriKey\ A \in used\ s$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def image-def*)

proposition *owner-iii-used*:
$s_0 \models s \Longrightarrow$
$(Owner\ n,\ \{\!|Num\ 3,\ PubKey\ A|\!\}) \in s \longrightarrow$
$PriKey\ A \in used\ s$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def image-def*)

proposition *asset-iii-used*:
$s_0 \models s \Longrightarrow$
$(Asset\ n,\ \{\!|Num\ 4,\ PubKey\ A|\!\}) \in s \longrightarrow$
$PriKey\ A \in used\ s$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def image-def*)

proposition *asset-i-unique* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (PriKey\ A)) \in s \longrightarrow$
$(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ A)) \in s \longrightarrow$
$m = n$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *frule asset-i-used* [*of* - *m A*],
drule asset-i-used [*of* - *n A*], *auto simp add*: *rel-def*)

proposition *owner-ii-unique* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Owner\ m,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \longrightarrow$
$(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \longrightarrow$
$m = n$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *frule owner-ii-used* [*of* - *m A*],
drule owner-ii-used [*of* - *n A*], *auto simp add*: *rel-def*)

proposition *asset-ii-unique* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ m,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
$(Asset\ n,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
$m = n$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *frule asset-ii-used* [*of* - *m A*],
drule asset-ii-used [*of* - *n A*], *auto simp add*: *rel-def*)

proposition *auth-prikey-asset-i* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (Auth\text{-}PriKey\ n)) \in s \longrightarrow$
$False$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *drule auth-prikey-used* [*of - n*],
auto simp add: *rel-def*)

proposition *auth-pubkey-owner-ii* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Owner\ m,\ \{\!|Num\ 1,\ Auth\text{-}PubKey\ n|\!\}) \in s \longrightarrow$
$False$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *drule auth-prikey-used* [*of - n*],
auto simp add: *rel-def*)

proposition *auth-pubkey-asset-ii* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ m,\ \{\!|Num\ 2,\ Auth\text{-}PubKey\ n|\!\}) \in s \longrightarrow$
$False$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *drule auth-prikey-used* [*of - n*],
auto simp add: *rel-def*)

proposition *asset-i-owner-ii* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (PriKey\ A)) \in s \longrightarrow$
$(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \longrightarrow$
$False$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *frule asset-i-used* [*of - m A*],
drule owner-ii-used [*of - n A*], *auto simp add*: *rel-def*)

proposition *asset-i-asset-ii* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (PriKey\ A)) \in s \longrightarrow$
$(Asset\ n,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
$False$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *frule asset-i-used* [*of - m A*],
drule asset-ii-used [*of - n A*], *auto simp add*: *rel-def*)

proposition *asset-ii-owner-ii* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ m,\ \{\!|Num\ 2,\ PubKey\ A|\!\}) \in s \longrightarrow$
$(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \longrightarrow$
$False$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *frule asset-ii-used* [*of - m A*],
drule owner-ii-used [*of - n A*], *auto simp add*: *rel-def*)

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

$(Owner\ n,\ \{\!|Num\ 3,\ PubKey\ A|\!\}) \in s \longrightarrow$
$False$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *frule asset-iii-used* [*of* - *m A*],
drule owner-iii-used [*of* - *n A*], *auto simp add*: *rel-def*)

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

proposition *owner-v-state* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s \longrightarrow$
$(Owner\ n,\ SesKey\ SK) \in s \wedge$
$(\exists A\ B\ C.\ Token\ n\ A\ B\ C\ SK \in used\ s \wedge B \in fst\ (snd\ SK))$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def*, *blast*)

proposition *asset-v-state* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0)) \in s \longrightarrow$
$(Asset\ n,\ SesKey\ SK) \in s \wedge Crypt\ (SesK\ SK)\ (Pwd\ n) \in used\ s$
by (*erule rtrancl-induct*, *simp-all add*: *rel-def image-def*,
((*erule disjE*)?, (*erule exE*)+, *simp add*: *Range-Un-eq*)+)

lemma *owner-seskey-nonce-1*:
$[\![s \vdash s';$
$(Owner\ n,\ SesKey\ SK) \in s \longrightarrow$
$(\exists S.\ fst\ SK = Some\ S \wedge Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S) \in used\ s) \vee$
$fst\ SK = None;$
$(Owner\ n,\ SesKey\ SK) \in s']\!] \Longrightarrow$
$(\exists S.\ fst\ SK = Some\ S \wedge Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S) \in used\ s') \vee$
$fst\ SK = None$
by (*simp add*: *rel-def*, (*erule disjE*, (*erule exE*)+, *simp*+)+,
split if-split-asm, *auto*)

proposition *owner-seskey-nonce* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Owner\ n,\ SesKey\ SK) \in s \longrightarrow$
$(\exists S.\ fst\ SK = Some\ S \wedge Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S) \in used\ s) \vee$
$fst\ SK = None$
by (*erule rtrancl-induct*, *simp add*: *image-def*, *rule impI*, *rule owner-seskey-nonce-1*)

proposition *owner-seskey-other* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Owner\ n,\ SesKey\ SK) \in s \longrightarrow$
$(\exists A\ B\ C\ D.\ fst\ (snd\ SK) = \{A,\ B\} \wedge snd\ (snd\ SK) = \{C,\ D\} \wedge$

$(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\})\in s\ \wedge$
$(Owner\ n,\ \{\!|Num\ 3,\ PubKey\ C|\!\})\in s\ \wedge$
$(Owner\ n,\ Crypt\ (SesK\ SK)\ (PubKey\ D))\in s)$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def*, *blast*+)

proposition *asset-seskey-nonce* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ n,\ SesKey\ SK)\in s \longrightarrow$
$(\exists S.\ fst\ SK = Some\ S \wedge (Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S))\in s)$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def*)

proposition *asset-seskey-other* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$(Asset\ n,\ SesKey\ SK)\in s \longrightarrow$
$(\exists A\ B\ C\ D.\ fst\ (snd\ SK) = \{A,\ B\} \wedge snd\ (snd\ SK) = \{C,\ D\} \wedge$
$(Asset\ n,\ \{\!|Num\ 2,\ PubKey\ B|\!\})\in s \wedge (Asset\ n,\ \{\!|Num\ 4,\ PubKey\ D|\!\})\in s\ \wedge$
$(Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK)\in s)$
by (*erule rtrancl-induct*, *auto simp add*: *rel-def*, *blast*)

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

proposition *used-prod* [*simp*]:
$A \neq \{\} \Longrightarrow used\ (A \times H) = H$
by (*simp add*: *Range-snd*)

proposition *parts-idem* [*simp*]:
$parts\ (parts\ H) = parts\ H$
by (*rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-mono*:
$H \subseteq H' \Longrightarrow parts\ H \subseteq parts\ H'$
by (*rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-msg-mono*:
$X \in H \Longrightarrow parts\text{-}msg\ X \subseteq parts\ H$
by (*subgoal-tac* $\{X\} \subseteq H$, *subst parts-msg-def*, *erule parts-mono*, *simp*)

lemma *parts-union-1*:
$parts\ (H \cup H') \subseteq parts\ H \cup parts\ H'$
by (*rule subsetI*, *erule parts.induct*, *auto*)

lemma *parts-union-2*:
$parts\ H \cup parts\ H' \subseteq parts\ (H \cup H')$
by (*rule subsetI*, *erule UnE*, *erule-tac* [!] *parts.induct*, *auto*)

proposition *parts-union* [*simp*]:
$parts\ (H \cup H') = parts\ H \cup parts\ H'$
by (*rule equalityI*, *rule parts-union-1*, *rule parts-union-2*)

proposition *parts-insert*:
$parts\ (insert\ X\ H) = parts\text{-}msg\ X \cup parts\ H$
by (*simp only*: *insert-def parts-union*, *subst parts-msg-def*, *simp*)

proposition *parts-msg-num* [*simp*]:
$parts\text{-}msg\ (Num\ n) = \{Num\ n\}$
by (*subst parts-msg-def*, *rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-msg-pwd* [*simp*]:
$parts\text{-}msg\ (Pwd\ n) = \{Pwd\ n\}$
by (*subst parts-msg-def*, *rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-msg-key* [*simp*]:
$parts\text{-}msg\ (Key\ K) = \{Key\ K\}$
by (*subst parts-msg-def*, *rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-msg-mult* [*simp*]:
$parts\text{-}msg\ (A \otimes B) = \{A \otimes B\}$
by (*subst parts-msg-def*, *rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-msg-hash* [*simp*]:
$parts\text{-}msg\ (Hash\ X) = \{Hash\ X\}$
by (*subst parts-msg-def*, *rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

lemma *parts-crypt-1*:
$parts\ \{Crypt\ K\ X\} \subseteq insert\ (Crypt\ K\ X)\ (parts\ \{X\})$
by (*rule subsetI*, *erule parts.induct*, *auto*)

lemma *parts-crypt-2*:
$insert\ (Crypt\ K\ X)\ (parts\ \{X\}) \subseteq parts\ \{Crypt\ K\ X\}$
by (*rule subsetI*, *simp*, *erule disjE*, *blast*, *erule parts.induct*, *auto*)

proposition *parts-msg-crypt* [*simp*]:
$parts\text{-}msg\ (Crypt\ K\ X) = insert\ (Crypt\ K\ X)\ (parts\text{-}msg\ X)$
by (*simp add*: *parts-msg-def*, *rule equalityI*, *rule parts-crypt-1*, *rule parts-crypt-2*)

lemma *parts-mpair-1*:
$parts\ \{\{\!|X,\ Y|\!\}\} \subseteq insert\ \{\!|X,\ Y|\!\}\ (parts\ \{X\} \cup parts\ \{Y\})$
by (*rule subsetI*, *erule parts.induct*, *auto*)

lemma *parts-mpair-2*:
$insert\ \{\!|X,\ Y|\!\}\ (parts\ \{X\} \cup parts\ \{Y\}) \subseteq parts\ \{\{\!|X,\ Y|\!\}\}$
by (*rule subsetI*, *simp*, *erule disjE*, *blast*, *erule disjE*, *erule-tac* [!] *parts.induct*, *auto*)

proposition *parts-msg-mpair* [*simp*]:
$parts\text{-}msg\ \{\!|X,\ Y|\!\} = insert\ \{\!|X,\ Y|\!\}\ (parts\text{-}msg\ X \cup parts\text{-}msg\ Y)$
by (*simp add*: *parts-msg-def*, *rule equalityI*, *rule parts-mpair-1*, *rule parts-mpair-2*)

proposition *parts-msg-idinfo* [*simp*]:
$parts\text{-}msg\ \langle n,\ X\rangle = \{\langle n,\ X\rangle\}$
by (*subst parts-msg-def*, *rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-msg-trace* [*simp*]:
$parts\text{-}msg\ (Log\ X) = \{Log\ X\}$
by (*subst parts-msg-def*, *rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-idinfo* [*simp*]:
$parts\ (IDInfo\ n\ `\ H) = IDInfo\ n\ `\ H$
by (*rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-trace* [*simp*]:
$parts\ (Log\ `\ H) = Log\ `\ H$
by (*rule equalityI*, *rule subsetI*, *erule parts.induct*, *auto*)

proposition *parts-dec*:
$[\![s' = insert\ (Spy,\ X)\ s \wedge (Spy,\ Crypt\ K\ X) \in s \wedge (Spy,\ Key\ (InvK\ K)) \in s;$
$\quad Y \in parts\text{-}msg\ X]\!] \Longrightarrow$
$\ Y \in parts\ (used\ s)$
by (*subgoal-tac* $X \in parts\ (used\ s)$, *drule parts-msg-mono* [*of X*], *auto*)

proposition *parts-enc*:
$[\![s' = insert\ (Spy,\ Crypt\ K\ X)\ s \wedge (Spy,\ X) \in s \wedge (Spy,\ Key\ K) \in s;$
$\quad Y \in parts\text{-}msg\ X]\!] \Longrightarrow$
$\ Y \in parts\ (used\ s)$
by (*subgoal-tac* $X \in parts\ (used\ s)$, *drule parts-msg-mono* [*of X*], *auto*)

proposition *parts-sep*:
$[\![s' = insert\ (Spy,\ X)\ (insert\ (Spy,\ Y)\ s) \wedge (Spy,\ \{\!|X,\ Y|\!\}) \in s;$
$\quad Z \in parts\text{-}msg\ X \vee Z \in parts\text{-}msg\ Y]\!] \Longrightarrow$
$\ Z \in parts\ (used\ s)$
by (*erule disjE*, *subgoal-tac* $X \in parts\ (used\ s)$, *drule parts-msg-mono* [*of X*],
subgoal-tac [*3*] $Y \in parts\ (used\ s)$, *drule-tac* [*3*] *parts-msg-mono* [*of Y*], *auto*)

proposition *parts-con*:
$[\![s' = insert\ (Spy,\ \{\!|X,\ Y|\!\})\ s \wedge (Spy,\ X) \in s \wedge (Spy,\ Y) \in s;$
$\quad Z \in parts\text{-}msg\ X \vee Z \in parts\text{-}msg\ Y]\!] \Longrightarrow$
$\ Z \in parts\ (used\ s)$
by (*erule disjE*, *subgoal-tac* $X \in parts\ (used\ s)$, *drule parts-msg-mono* [*of X*],
subgoal-tac [*3*] $Y \in parts\ (used\ s)$, *drule-tac* [*3*] *parts-msg-mono* [*of Y*], *auto*)

lemma *parts-init-1*:
$parts\ (used\ s_0) \subseteq used\ s_0 \cup range\ (Hash \circ Agent) \cup$
$\quad range\ (Hash \circ Auth\text{-}PubKey) \cup$
$\quad range\ (\lambda n.\ \{\!|Hash\ (Agent\ n),\ Hash\ (Auth\text{-}PubKey\ n)|\!\})$
by (*rule subsetI*, *erule parts.induct*, (*clarify* | *simp add*: *Range-iff image-def*)+)

lemma *parts-init-2*:
$used\ s_0 \cup range\ (Hash \circ Agent) \cup range\ (Hash \circ Auth\text{-}PubKey) \cup$
$range\ (\lambda n.\ \{\!|Hash\ (Agent\ n),\ Hash\ (Auth\text{-}PubKey\ n)|\!\}) \subseteq parts\ (used\ s_0)$
by (*rule subsetI*, *auto simp add*: *parts-insert*)

proposition *parts-init*:
$parts\ (used\ s_0) = used\ s_0 \cup range\ (Hash \circ Agent) \cup$
$range\ (Hash \circ Auth\text{-}PubKey) \cup$
$range\ (\lambda n.\ \{\!|Hash\ (Agent\ n),\ Hash\ (Auth\text{-}PubKey\ n)|\!\})$
by (*rule equalityI*, *rule parts-init-1*, *rule parts-init-2*)

proposition *parts-crypt-prikey-start*:
$[\![s \vdash s';\ Crypt\ K\ (PriKey\ A) \in parts\ (used\ s');$
$Crypt\ K\ (PriKey\ A) \notin parts\ (used\ s)]\!] \Longrightarrow$
$(\exists\, n.\ K = Auth\text{-}ShaKey\ n\ \wedge$
$(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ A)) \in s') \vee$
$\{PriKey\ A,\ Key\ K\} \subseteq spied\ s'$
by (*simp add*: *rel-def*, *erule disjE*, (*erule exE*)+, *simp add*: *parts-insert*, *blast*,
(((*erule disjE*)?, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
((*drule parts-dec* | *erule disjE*, *simp*, *drule parts-enc* |
drule parts-sep | *drule parts-con*), *simp*+)?)+)

proposition *parts-crypt-prikey*:
$[\![s_0 \models s;\ Crypt\ K\ (PriKey\ A) \in parts\ (used\ s)]\!] \Longrightarrow$
$(\exists\, n.\ K = Auth\text{-}ShaKey\ n\ \wedge$
$(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ A)) \in s) \vee$
$\{PriKey\ A,\ Key\ K\} \subseteq spied\ s$
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*,
(*erule exE*)+, (*erule conjE*)+, *frule parts-crypt-prikey-start*, *assumption*+,
(*drule state-subset*)+, *blast*)

proposition *parts-crypt-pubkey-start*:
$[\![s \vdash s';\ Crypt\ (SesK\ SK)\ (PubKey\ C) \in parts\ (used\ s');$
$Crypt\ (SesK\ SK)\ (PubKey\ C) \notin parts\ (used\ s)]\!] \Longrightarrow$
$C \in snd\ (snd\ SK) \wedge ((\exists\, n.\ (Owner\ n,\ SesKey\ SK) \in s') \vee$
$(\exists\, n\ B.\ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s')) \vee$
$SesKey\ SK \in spied\ s'$
by (*simp add*: *rel-def*, (*erule disjE*, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
blast, *erule disjE*, (*erule exE*)+, *simp add*: *parts-insert image-iff*, *blast*,
(((*erule disjE*)?, ((*erule exE*)+)?, *simp add*: *parts-insert image-iff*)+,
((*drule parts-dec* | *drule parts-enc* | *drule parts-sep* | *drule parts-con*), *simp*+)?)+)

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 $\in$ *spied s*
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*, (*erule exE*)+, (*erule conjE*)+, *frule parts-crypt-pubkey-start*, *assumption*+, (*drule state-subset*)+, *blast*)

proposition *parts-crypt-key-start*:
$[\![s \vdash s'$; *Crypt K* (*Key K'*) $\in$ *parts* (*used s'*);
Crypt K (*Key K'*) $\notin$ *parts* (*used s*); $K' \notin$ *range PriK* $\cup$ *range PubK*$]\!] \Longrightarrow$
$\{$*Key K'*, *Key K*$\}$ $\subseteq$ *spied s'*
by (*simp add*: *rel-def*, (((*erule disjE*)?, ((*erule exE*)+)?, *simp add*: *parts-insert image-iff*)+, ((*drule parts-dec* | *drule parts-enc* | *drule parts-sep* | *drule parts-con*), *simp*+)?)+)

proposition *parts-crypt-key*:
$[\![s_0 \models s$; *Crypt K* (*Key K'*) $\in$ *parts* (*used s*);
$K' \notin$ *range PriK* $\cup$ *range PubK*$]\!] \Longrightarrow$
$\{$*Key K'*, *Key K*$\}$ $\subseteq$ *spied s*
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*, (*erule exE*)+, (*erule conjE*)+, *frule parts-crypt-key-start*, *assumption*+, (*drule state-subset*)+, *blast*)

proposition *parts-crypt-sign-start*:
$[\![s \vdash s'$; *Crypt* (*SesK SK*) (*Sign n A*) $\in$ *parts* (*used s'*);
Crypt (*SesK SK*) (*Sign n A*) $\notin$ *parts* (*used s*)$]\!] \Longrightarrow$
(*Asset n*, *SesKey SK*) $\in s' \vee$ *SesKey SK* $\in$ *spied s'*
by (*simp add*: *rel-def*, (((*erule disjE*)?, ((*erule exE*)+)?, *simp add*: *parts-insert image-iff*)+, ((*drule parts-dec* | *drule parts-enc* | *drule parts-sep* | *drule parts-con*), *simp*+)?)+)

proposition *parts-crypt-sign*:
$[\![s_0 \models s$; *Crypt* (*SesK SK*) (*Sign n A*) $\in$ *parts* (*used s*)$]\!] \Longrightarrow$
(*Asset n*, *SesKey SK*) $\in s \vee$ *SesKey SK* $\in$ *spied s*
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*, (*erule exE*)+, (*erule conjE*)+, *frule parts-crypt-sign-start*, *assumption*+, (*drule state-subset*)+, *blast*)

proposition *parts-crypt-pwd-start*:
$[\![s \vdash s'$; *Crypt K* (*Pwd n*) $\in$ *parts* (*used s'*);
Crypt K (*Pwd n*) $\notin$ *parts* (*used s*)$]\!] \Longrightarrow$
($\exists$ *SK*. $K =$ *SesK SK* $\wedge$ (*Owner n*, *Crypt* (*SesK SK*) (*Pwd n*)) $\in s'$) $\vee$
$\{$*Pwd n*, *Key K*$\}$ $\subseteq$ *spied s'*
by (*simp add*: *rel-def*, (((*erule disjE*)?, ((*erule exE*)+)?, *simp add*: *parts-insert image-iff*)+, ((*drule parts-dec* | *drule parts-enc* | *drule parts-sep* | *drule parts-con*), *simp*+)?)+)

proposition *parts-crypt-pwd*:

$[\![s_0 \models s;\ \mathit{Crypt}\ K\ (\mathit{Pwd}\ n) \in \mathit{parts}\ (\mathit{used}\ s)]\!] \Longrightarrow$
$(\exists\, \mathit{SK}.\ K = \mathit{SesK}\ \mathit{SK} \wedge (\mathit{Owner}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Pwd}\ n)) \in s) \vee$
$\{\mathit{Pwd}\ n,\ \mathit{Key}\ K\} \subseteq \mathit{spied}\ s$
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*,
(*erule exE*)+, (*erule conjE*)+, *frule parts-crypt-pwd-start*, *assumption*+,
(*drule state-subset*)+, *blast*)

proposition *parts-crypt-num-start*:
$[\![s \vdash s';\ \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Num}\ 0) \in \mathit{parts}\ (\mathit{used}\ s');$
$\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Num}\ 0) \notin \mathit{parts}\ (\mathit{used}\ s)]\!] \Longrightarrow$
$(\exists\, n.\ (\mathit{Asset}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Num}\ 0)) \in s') \vee \mathit{SesKey}\ \mathit{SK} \in \mathit{spied}\ s'$
by (*simp add*: *rel-def*, (*erule disjE*, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
blast, (((*erule disjE*)?, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
((*drule parts-dec* | *erule disjE*, *simp*, *drule parts-enc* |
drule parts-sep | *drule parts-con*), *simp*+)?)+)

proposition *parts-crypt-num*:
$[\![s_0 \models s;\ \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Num}\ 0) \in \mathit{parts}\ (\mathit{used}\ s)]\!] \Longrightarrow$
$(\exists\, n.\ (\mathit{Asset}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{Num}\ 0)) \in s) \vee \mathit{SesKey}\ \mathit{SK} \in \mathit{spied}\ s$
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*,
(*erule exE*)+, (*erule conjE*)+, *frule parts-crypt-num-start*, *assumption*+,
(*drule state-subset*)+, *blast*)

proposition *parts-crypt-mult-start*:
$[\![s \vdash s';\ \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (A \otimes B) \in \mathit{parts}\ (\mathit{used}\ s');$
$\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (A \otimes B) \notin \mathit{parts}\ (\mathit{used}\ s)]\!] \Longrightarrow$
$B \in \mathit{fst}\ (\mathit{snd}\ \mathit{SK}) \wedge (\exists\, n\ C.\ (\mathit{Asset}\ n,\ \mathit{Token}\ n\ (\mathit{Auth\text{-}PriK}\ n)\ B\ C\ \mathit{SK}) \in s') \vee$
$\{A \otimes B,\ \mathit{SesKey}\ \mathit{SK}\} \subseteq \mathit{spied}\ s$
by (*simp add*: *rel-def*, (*erule disjE*, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
blast, (((*erule disjE*)?, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
((*drule parts-dec* | *erule disjE*, *simp*, *drule parts-enc* |
drule parts-sep | *drule parts-con*), *simp*+)?)+)

proposition *parts-crypt-mult*:
$[\![s_0 \models s;\ \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (A \otimes B) \in \mathit{parts}\ (\mathit{used}\ s)]\!] \Longrightarrow$
$B \in \mathit{fst}\ (\mathit{snd}\ \mathit{SK}) \wedge (\exists\, n\ C.\ (\mathit{Asset}\ n,\ \mathit{Token}\ n\ (\mathit{Auth\text{-}PriK}\ n)\ B\ C\ \mathit{SK}) \in s) \vee$
$\{A \otimes B,\ \mathit{SesKey}\ \mathit{SK}\} \subseteq \mathit{spied}\ s$
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*,
(*erule exE*)+, (*erule conjE*)+, *frule parts-crypt-mult-start*, *assumption*+,
drule converse-rtrancl-into-rtrancl, *assumption*, *drule state-subset* [*of* - *s*],
drule spied-subset [*of* - *s*], *blast*)

proposition *parts-mult-start*:
$[\![s \vdash s';\ A \otimes B \in \mathit{parts}\ (\mathit{used}\ s');\ A \otimes B \notin \mathit{parts}\ (\mathit{used}\ s)]\!] \Longrightarrow$

$(\exists\, n\ SK.\ A = \textit{Auth-PriK}\ n \wedge (\textit{Asset}\ n,\ \{\!|\textit{Num}\ 2,\ \textit{PubKey}\ B|\!\}) \in s' \wedge$
$\textit{Crypt}\ (\textit{SesK}\ SK)\ (A \otimes B) \in \textit{parts}\ (\textit{used}\ s')) \vee$
$\{\textit{PriKey}\ A,\ \textit{PriKey}\ B\} \subseteq \textit{spied}\ s'$

by (*simp add*: *rel-def*, (*erule disjE*, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
blast, (((*erule disjE*)?, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
((*drule parts-dec* | *drule parts-enc* | *drule parts-sep* | *drule parts-con*), *simp*+)?)+)

proposition *parts-mult*:
$[\![s_0 \models s;\ A \otimes B \in \textit{parts}\ (\textit{used}\ s)]\!] \Longrightarrow$
$(\exists\, n.\ A = \textit{Auth-PriK}\ n \wedge (\textit{Asset}\ n,\ \{\!|\textit{Num}\ 2,\ \textit{PubKey}\ B|\!\}) \in s) \vee$
$\{\textit{PriKey}\ A,\ \textit{PriKey}\ B\} \subseteq \textit{spied}\ s$
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*,
(*erule exE*)+, (*erule conjE*)+, *frule parts-mult-start*, *assumption*+,
(*drule state-subset*)+, *blast*)

proposition *parts-mpair-key-start*:
$[\![s \vdash s';\ \{\!|X,\ Y|\!\} \in \textit{parts}\ (\textit{used}\ s');\ \{\!|X,\ Y|\!\} \notin \textit{parts}\ (\textit{used}\ s);$
$X = \textit{Key}\ K \vee Y = \textit{Key}\ K \wedge K \notin \textit{range PubK}]\!] \Longrightarrow$
$\{X,\ Y\} \subseteq \textit{spied}\ s'$
by (*erule disjE*, *simp-all add*: *rel-def*,
(((*erule disjE*)?, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
((*drule parts-dec* | *drule parts-enc* |
drule parts-sep | *erule disjE*, *simp*, *drule parts-con*), *simp*+)?)+)

proposition *parts-mpair-key*:
$[\![s_0 \models s;\ \{\!|X,\ Y|\!\} \in \textit{parts}\ (\textit{used}\ s);$
$X = \textit{Key}\ K \vee Y = \textit{Key}\ K \wedge K \notin \textit{range PubK}]\!] \Longrightarrow$
$\{X,\ Y\} \subseteq \textit{spied}\ s$
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*,
blast, (*erule exE*)+, (*erule conjE*)+, *frule parts-mpair-key-start*, *assumption*+,
(*drule state-subset*)+, *blast*)

proposition *parts-mpair-pwd-start*:
$[\![s \vdash s';\ \{\!|X,\ Y|\!\} \in \textit{parts}\ (\textit{used}\ s');\ \{\!|X,\ Y|\!\} \notin \textit{parts}\ (\textit{used}\ s);$
$X = \textit{Pwd}\ n \vee Y = \textit{Pwd}\ n]\!] \Longrightarrow$
$\{X,\ Y\} \subseteq \textit{spied}\ s'$
by (*erule disjE*, *simp-all add*: *rel-def*,
(((*erule disjE*)?, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
((*drule parts-dec* | *drule parts-enc* |
drule parts-sep | *erule disjE*, *simp*, *drule parts-con*), *simp*+)?)+)

proposition *parts-mpair-pwd*:
$[\![s_0 \models s;\ \{\!|X,\ Y|\!\} \in \textit{parts}\ (\textit{used}\ s);\ X = \textit{Pwd}\ n \vee Y = \textit{Pwd}\ n]\!] \Longrightarrow$
$\{X,\ Y\} \subseteq \textit{spied}\ s$
by (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*,
blast, (*erule exE*)+, (*erule conjE*)+, *frule parts-mpair-pwd-start*, *assumption*+,

$(drule\ state\text{-}subset)+,\ blast)$

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*
proof −
have $C \in snd\ (snd\ SK) \wedge ((\exists\, n.\ (Owner\ n,\ SesKey\ SK) \in s') \vee$
$(\exists\, n\ B.\ (Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s')) \vee$
$SesKey\ SK \in spied\ s'$
(**is** $?P\ C \wedge ((\exists\, n.\ ?Q\ n\ s') \vee (\exists\, n\ B.\ ?R\ n\ B\ C\ s')) \vee\ ?S\ s'$)
by $(rule\ parts\text{-}crypt\text{-}pubkey\text{-}start\ [OF\ B\ C\ D])$
then obtain $n\ B$ **where** $?P\ C$ **and** $?R\ n\ B\ C\ s'$
using E **and** F **by** *blast*
moreover have $\neg\ ?R\ n\ B\ C\ s$
using D **by** *blast*
ultimately have $\exists\, D.\ Crypt\ (SesK\ SK)\ (PubKey\ D) \in used\ s$
(**is** $\exists\, D.\ ?U\ D$)
using B **by** $(auto\ simp\ add\text{: } rel\text{-}def)$
then obtain D **where** $?U\ D$ **..**
hence $?P\ D \wedge ((\exists\, n.\ ?Q\ n\ s) \vee (\exists\, n\ B.\ ?R\ n\ B\ D\ s)) \vee\ ?S\ s$
by $(rule\text{-}tac\ parts\text{-}crypt\text{-}pubkey\ [OF\ A],\ blast)$
moreover have G: $s \subseteq s'$
by $(rule\ state\text{-}subset,\ insert\ B,\ simp)$
have $\forall\, n.\ (Owner\ n,\ SesKey\ SK) \notin s$
by $(rule\ allI,\ rule\ notI,\ drule\ subsetD\ [OF\ G],\ insert\ E,\ simp)$
moreover have H: $spied\ s \subseteq spied\ s'$
by $(rule\ Image\text{-}mono\ [OF\ G],\ simp)$
have $SesKey\ SK \notin spied\ s$
by $(rule\ notI,\ drule\ subsetD\ [OF\ H],\ insert\ F,\ contradiction)$
ultimately obtain $n'\ B'$ **where** $?R\ n'\ B'\ D\ s$ **by** *blast*
have $\exists\, A'\ D'.\ fst\ (snd\ SK) = \{A',\ B'\} \wedge snd\ (snd\ SK) = \{D,\ D'\} \wedge$
$(Asset\ n',\ \{\!|Num\ 2,\ PubKey\ B'|\!\}) \in s \wedge$
$(Asset\ n',\ \{\!|Num\ 4,\ PubKey\ D'|\!\}) \in s \wedge$
$?U\ D' \wedge (Asset\ n',\ PubKey\ B') \in s$
by $(rule\ asset\text{-}iv\text{-}state\ [OF\ A\ \langle ?R\ n'\ B'\ D\ s\rangle])$
then obtain D' **where** $snd\ (snd\ SK) = \{D,\ D'\}$ **and** $?U\ D'$ **by** *blast*
hence $Crypt\ (SesK\ SK)\ (PubKey\ C) \in parts\ (used\ s)$
using $\langle ?P\ C\rangle$ **and** $\langle ?U\ D\rangle$ **by** *auto*
thus *False*
using D **by** *contradiction*
qed

proposition *parts-pubkey-false*:
$[\![s_0 \models s;$ *Crypt* (*SesK SK*) (*PubKey C*) $\in$ *parts* (*used s*);
$\forall\, n.$ (*Owner n*, *SesKey SK*) $\notin$ *s*; *SesKey SK* $\notin$ *spied s*$]\!] \Longrightarrow$
False
proof (*drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*,
(*erule exE*)+, (*erule conjE*)+, *erule parts-pubkey-false-start*, *assumption*+,
rule allI, *rule-tac* [!] *notI*)
fix *v n*
assume (*Owner n*, *SesKey SK*) $\in$ *v* **and** $v \models s$
hence (*Owner n*, *SesKey SK*) $\in$ *s*
by (*erule-tac rev-subsetD*, *rule-tac state-subset*)
moreover assume $\forall\, n.$ (*Owner n*, *SesKey SK*) $\notin$ *s*
ultimately show *False* **by** *simp*
next
fix *v*
assume *SesKey SK* $\in$ *spied v* **and** $v \models s$
hence *SesKey SK* $\in$ *spied s*
by (*erule-tac rev-subsetD*, *rule-tac spied-subset*)
moreover assume *SesKey SK* $\notin$ *spied s*
ultimately show *False* **by** *contradiction*
qed

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-PriKey n* $\in$ *spied s* $\wedge$
($\exists$ *C SK*. (*Asset n*, *Token n* (*Auth-PriK n*) *B C SK*) $\in$ *s*)
(**is** - $\wedge$ ($\exists$ *C SK*. *?P n C SK s*))
proof $-$
have $\exists\, A.$ ($A \otimes B \in$ *spied s* $\vee$ $B \otimes A \in$ *spied s*) $\wedge$ *PriKey A* $\in$ *spied s*
proof (*insert B C D*, *auto simp add*: *rel-def*, *rule-tac* [!] *FalseE*)
assume *Key* (*PriK B*) $\notin$ *used s*
moreover have *PriKey B* $\in$ *used s*
by (*rule asset-ii-used* [*OF A*, *THEN mp*, *OF E*])
ultimately show *False* **by** *simp*
next
fix *K*
assume (*Spy*, *Crypt K* (*Key* (*PriK B*))) $\in$ *s*
hence *Crypt K* (*PriKey B*) $\in$ *parts* (*used s*) **by** *auto*
hence ($\exists\, m.$ *K* = *Auth-ShaKey m* $\wedge$
(*Asset m*, *Crypt* (*Auth-ShaKey m*) (*PriKey B*)) $\in$ *s*) $\vee$
$\{$*PriKey B*, *Key K*$\}$ $\subseteq$ *spied s*
(**is** ($\exists\, m.$ - $\wedge$ *?P m*) $\vee$ -)

    by (*rule parts-crypt-prikey* [*OF A*])
  then obtain *m* **where** *?P m*
    using *D* **by** *blast*
  thus *False*
    by (*rule asset-i-asset-ii* [*OF A* - *E*])
next
  fix *Y*
  assume (*Spy*, $\{\!|$*Key* (*PriK B*), *Y*$|\!\}$) $\in$ *s*
  hence $\{\!|$*PriKey B*, *Y*$|\!\}$ $\in$ *parts* (*used s*) **by** *auto*
  hence {*PriKey B*, *Y*} $\subseteq$ *spied s*
    by (*rule parts-mpair-key* [*OF A*, **where** *K* = *PriK B*], *simp*)
  thus *False*
    using *D* **by** *simp*
next
  fix *X*
  assume (*Spy*, $\{\!|$*X*, *Key* (*PriK B*)$|\!\}$) $\in$ *s*
  hence $\{\!|$*X*, *PriKey B*$|\!\}$ $\in$ *parts* (*used s*) **by** *auto*
  hence {*X*, *PriKey B*} $\subseteq$ *spied s*
    by (*rule parts-mpair-key* [*OF A*, **where** *K* = *PriK B*], *simp add*: *image-def*)
  thus *False*
    using *D* **by** *simp*
qed
then obtain *A* **where** *F*: *PriKey A* $\in$ *spied s* **and**
  A $\otimes$ *B* $\in$ *spied s* $\vee$ *B* $\otimes$ *A* $\in$ *spied s*
  by *blast*
hence *A* $\otimes$ *B* $\in$ *parts* (*used s*) $\vee$ *B* $\otimes$ *A* $\in$ *parts* (*used s*) **by** *blast*
moreover have *B* $\otimes$ *A* $\notin$ *parts* (*used s*)
proof
  assume *B* $\otimes$ *A* $\in$ *parts* (*used s*)
  hence ($\exists$ *m*. *B* = *Auth-PriK m* $\wedge$ (*Asset m*, $\{\!|$*Num 2*, *PubKey A*$|\!\}$) $\in$ *s*) $\vee$
    {*PriKey B*, *PriKey A*} $\subseteq$ *spied s*
    by (*rule parts-mult* [*OF A*])
  then obtain *m* **where** *B* = *Auth-PriK m*
    using *D* **by** *blast*
  hence (*Asset n*, $\{\!|$*Num 2*, *Auth-PubKey m*$|\!\}$) $\in$ *s*
    using *E* **by** *simp*
  thus *False*
    by (*rule auth-pubkey-asset-ii* [*OF A*])
qed
ultimately have *A* $\otimes$ *B* $\in$ *parts* (*used s*) **by** *simp*
with *A* **have** $\exists$ *u v*. s_0 $\models$ *u* $\wedge$ *u* $\vdash$ *v* $\wedge$ *v* $\models$ *s* $\wedge$
  A $\otimes$ *B* $\notin$ *parts* (*used u*) $\wedge$ *A* $\otimes$ *B* $\in$ *parts* (*used v*)
  by (*rule rtrancl-start*, *subst parts-init*, *simp add*: *Range-iff image-def*)
then obtain *u v* **where** *G*: *u* $\vdash$ *v* **and** *H*: *v* $\models$ *s* **and**
  I: *A* $\otimes$ *B* $\notin$ *parts* (*used u*) **and** *A* $\otimes$ *B* $\in$ *parts* (*used v*)
  by *blast*
hence ($\exists$ *m SK*. *A* = *Auth-PriK m* $\wedge$ (*Asset m*, $\{\!|$*Num 2*, *PubKey B*$|\!\}$) $\in$ *v* $\wedge$
  Crypt (*SesK SK*) (*A* $\otimes$ *B*) $\in$ *parts* (*used v*)) $\vee$
  {*PriKey A*, *PriKey B*} $\subseteq$ *spied v*

```
    by (rule-tac parts-mult-start, simp-all)
  moreover have PriKey B ∉ spied v
  proof
    assume PriKey B ∈ spied v
    hence PriKey B ∈ spied s
      by (rule rev-subsetD, rule-tac spied-subset [OF H])
    thus False
      using D by contradiction
  qed
  ultimately obtain m SK where
    J: Crypt (SesK SK) (A ⊗ B) ∈ parts (used v) and
    A = Auth-PriK m and (Asset m, {|Num 2, PubKey B|}) ∈ v
    by blast
  moreover from this have (Asset m, {|Num 2, PubKey B|}) ∈ s
    by (erule-tac rev-subsetD, rule-tac state-subset [OF H])
  hence m = n
    by (rule asset-ii-unique [OF A - E])
  ultimately have K: Auth-PriKey n ∈ spied s
    using F by simp
  have Crypt (SesK SK) (A ⊗ B) ∉ parts (used u)
    using I by blast
  hence B ∈ fst (snd SK) ∧ (∃ m C. ?P m C SK v) ∨
    {A ⊗ B, SesKey SK} ⊆ spied u
    by (rule parts-crypt-mult-start [OF G J])
  moreover have A ⊗ B ∉ spied u
    using I by blast
  ultimately obtain m' C where ?P m' C SK v by blast
  hence ?P m' C SK s
    by (rule rev-subsetD, rule-tac state-subset [OF H])
  moreover from this have ∃ A' D. fst (snd SK) = {A', B} ∧
    snd (snd SK) = {C, D} ∧ (Asset m', {|Num 2, PubKey B|}) ∈ s ∧
    (Asset m', {|Num 4, PubKey D|}) ∈ s ∧
    Crypt (SesK SK) (PubKey D) ∈ used s ∧ (Asset m', PubKey B) ∈ s
    by (rule asset-iv-state [OF A])
  hence (Asset m', {|Num 2, PubKey B|}) ∈ s by blast
  hence m' = n
    by (rule asset-ii-unique [OF A - E])
  ultimately show ?thesis
    using K by blast
qed

proposition asset-ii-spied:
  assumes
    A: s0 |= s and
    B: PriKey B ∈ spied s and
    C: (Asset n, {|Num 2, PubKey B|}) ∈ s
  shows Auth-PriKey n ∈ spied s ∧
    (∃ C SK. (Asset n, Token n (Auth-PriK n) B C SK) ∈ s)
    (is ?P s)
```

proof –
  have $\exists\, u\ v.\ s_0 \models u \wedge u \vdash v \wedge v \models s \wedge$
    $(Asset\ n, \{\!|Num\ 2, PubKey\ B|\!\}) \notin u \wedge (Asset\ n, \{\!|Num\ 2, PubKey\ B|\!\}) \in v$
    using A **and** C **by** (*rule rtrancl-start*, *auto*)
  then obtain $u\ v$ **where** $v \models s$ **and** $(Asset\ n, \{\!|Num\ 2, PubKey\ B|\!\}) \notin u$ **and**
    D: $s_0 \models u$ **and** E: $u \vdash v$ **and** F: $(Asset\ n, \{\!|Num\ 2, PubKey\ B|\!\}) \in v$
    by *blast*
  moreover from *this* **have** $PriKey\ B \notin spied\ v$
    by (*auto simp add*: *rel-def*)
  ultimately have $\exists\, w\ x.\ v \models w \wedge w \vdash x \wedge x \models s \wedge$
    $PriKey\ B \notin spied\ w \wedge PriKey\ B \in spied\ x$
    using B **by** (*rule-tac rtrancl-start*, *simp-all*)
  then obtain $w\ x$ **where** $PriKey\ B \notin spied\ w$ **and** $PriKey\ B \in spied\ x$ **and**
    G: $v \models w$ **and** H: $w \vdash x$ **and** I: $x \models s$
    by *blast*
  moreover from *this* **have** $s_0 \models w$
    using D **and** E **by** *simp*
  moreover have $(Asset\ n, \{\!|Num\ 2, PubKey\ B|\!\}) \in w$
    by (*rule rev-subsetD* [*OF F*], *rule state-subset* [*OF G*])
  ultimately have *?P w*
    by (*rule-tac asset-ii-spied-start*, *simp-all*)
  moreover have $w \subseteq s$
    using H **and** I **by** (*rule-tac state-subset*, *simp*)
  ultimately show *?thesis* **by** *blast*
qed

proposition *asset-iv-unique*:
  assumes
    A: $s_0 \models s$ **and**
    B: $(Asset\ m, Token\ m\ (Auth\text{-}PriK\ m)\ B\ C'\ SK') \in s$ **and**
    C: $(Asset\ n, Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s$
      (**is** *?P n C SK s*)
  shows $m = n \wedge C' = C \wedge SK' = SK$
proof (*subst* (*2*) *cases-simp* [*of* $m = n$, *symmetric*], *simp*, *rule conjI*, *rule impI*, *rule ccontr*)
  assume D: $\neg\ (C' = C \wedge SK' = SK)$ **and** $m = n$
  moreover have $\exists\, u\ v.\ s_0 \models u \wedge u \vdash v \wedge v \models s \wedge$
    $\neg\ (?P\ m\ C'\ SK'\ u \wedge ?P\ n\ C\ SK\ u) \wedge ?P\ m\ C'\ SK'\ v \wedge ?P\ n\ C\ SK\ v$
    using B **and** C **by** (*rule-tac rtrancl-start* [*OF A*], *auto*)
  ultimately obtain $u\ v$ **where** E: $s_0 \models u$ **and** F: $u \vdash v$ **and**
    G: $?P\ n\ C'\ SK'\ v$ **and** H: $?P\ n\ C\ SK\ v$ **and**
    $\neg\ ?P\ n\ C'\ SK'\ u \vee \neg\ ?P\ n\ C\ SK\ u$
    by *blast*
  moreover {
    assume I: $\neg\ ?P\ n\ C'\ SK'\ u$
    hence $?P\ n\ C\ SK\ u$
      by (*insert D F G H*, *auto simp add*: *rel-def*)
    hence $\exists\, A\ D.\ fst\ (snd\ SK) = \{A, B\} \wedge snd\ (snd\ SK) = \{C, D\} \wedge$

$(Asset\ n,\ \{|Num\ 2,\ PubKey\ B|\}) \in u \wedge (Asset\ n,\ \{|Num\ 4,\ PubKey\ D|\}) \in u \wedge$
$Crypt\ (SesK\ SK)\ (PubKey\ D) \in used\ u \wedge (Asset\ n,\ PubKey\ B) \in u$
by (*rule asset-iv-state* [*OF E*])
moreover have $(Asset\ n,\ PubKey\ B) \notin u$
by (*insert F G I*, *auto simp add*: *rel-def*)
ultimately have *False* **by** *simp*
}
moreover {
assume I: $\neg\ ?P\ n\ C\ SK\ u$
hence $?P\ n\ C'\ SK'\ u$
by (*insert D F G H*, *auto simp add*: *rel-def*)
hence $\exists\ A\ D.\ fst\ (snd\ SK') = \{A,\ B\} \wedge snd\ (snd\ SK') = \{C',\ D\} \wedge$
$(Asset\ n,\ \{|Num\ 2,\ PubKey\ B|\}) \in u \wedge (Asset\ n,\ \{|Num\ 4,\ PubKey\ D|\}) \in u \wedge$
$Crypt\ (SesK\ SK')\ (PubKey\ D) \in used\ u \wedge (Asset\ n,\ PubKey\ B) \in u$
by (*rule asset-iv-state* [*OF E*])
moreover have $(Asset\ n,\ PubKey\ B) \notin u$
by (*insert F H I*, *auto simp add*: *rel-def*)
ultimately have *False* **by** *simp*
}
ultimately show *False* **by** *blast*
next
have $\exists\ A\ D.\ fst\ (snd\ SK') = \{A,\ B\} \wedge snd\ (snd\ SK') = \{C',\ D\} \wedge$
$(Asset\ m,\ \{|Num\ 2,\ PubKey\ B|\}) \in s \wedge (Asset\ m,\ \{|Num\ 4,\ PubKey\ D|\}) \in s \wedge$
$Crypt\ (SesK\ SK')\ (PubKey\ D) \in used\ s \wedge (Asset\ m,\ PubKey\ B) \in s$
(**is** $?Q\ m\ C'\ SK'$)
by (*rule asset-iv-state* [*OF A B*])
hence $(Asset\ m,\ \{|Num\ 2,\ PubKey\ B|\}) \in s$ **by** *blast*
moreover have $?Q\ n\ C\ SK$
by (*rule asset-iv-state* [*OF A C*])
hence $(Asset\ n,\ \{|Num\ 2,\ PubKey\ B|\}) \in s$ **by** *blast*
ultimately show $m = n$
by (*rule asset-ii-unique* [*OF A*])
qed

theorem *sigkey-secret*:
$s_0 \models s \Longrightarrow SigKey \notin spied\ s$
proof (*erule rtrancl-induct*, *simp add*: *image-def*)
fix $s\ s'$
assume
A: $s_0 \models s$ **and**
B: $s \vdash s'$ **and**
C: $SigKey \notin spied\ s$
show $SigKey \notin spied\ s'$
proof (*insert B C*, *auto simp add*: *rel-def*)
fix K
assume $(Spy,\ Crypt\ K\ SigKey) \in s$
hence $Crypt\ K\ SigKey \in parts\ (used\ s)$ **by** *blast*
hence $\{SigKey,\ Key\ K\} \subseteq spied\ s$

by (*rule parts-crypt-key* [*OF A*], *simp add*: *image-def*)
with *C* **show** *False* **by** *simp*
next
fix *Y*
assume (*Spy*, $\{\!|$*SigKey*, *Y*$|\!\}$) $\in$ *s*
hence $\{\!|$*SigKey*, *Y*$|\!\}$ $\in$ *parts* (*used s*) **by** *blast*
hence $\{$*SigKey*, *Y*$\}$ $\subseteq$ *spied s*
by (*rule parts-mpair-key* [*OF A*, **where** *K* = *SigK*], *simp*)
with *C* **show** *False* **by** *simp*
next
fix *X*
assume (*Spy*, $\{\!|$*X*, *SigKey*$|\!\}$) $\in$ *s*
hence $\{\!|$*X*, *SigKey*$|\!\}$ $\in$ *parts* (*used s*) **by** *blast*
hence $\{$*X*, *SigKey*$\}$ $\subseteq$ *spied s*
by (*rule parts-mpair-key* [*OF A*, **where** *K* = *SigK*], *simp add*: *image-def*)
with *C* **show** *False* **by** *simp*
qed
qed

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*)$]\!] \Longrightarrow$
A = *Auth-PriK n*
by (*simp add*: *rel-def*, (((*erule disjE*)?, (*erule exE*)+, *simp add*: *parts-insert image-iff*)+,
((*drule parts-dec* | *erule disjE*, *insert sigkey-secret* [*OF A*], *simp*, *drule parts-enc* | *drule parts-sep* | *drule parts-con*), *simp*+)?)+)

proposition *parts-sign*:
$[\![s_0 \models s$; *Sign n A* $\in$ *parts* (*used s*)$]\!] \Longrightarrow$
A = *Auth-PriK n*
by (*rule classical*, *drule rtrancl-start*, *assumption*, *subst parts-init*, *simp add*: *Range-iff image-def*, (*erule exE*)+, (*erule conjE*)+, *drule parts-sign-start*)

theorem *auth-shakey-secret*:
$[\![s_0 \models s$; *n* $\notin$ *bad-shakey*$]\!] \Longrightarrow$
Key (*Auth-ShaKey n*) $\notin$ *spied s*
proof (*erule rtrancl-induct*, *simp add*: *image-def*)
fix *s* s'
assume
A: $s_0 \models s$ **and**
B: $s \vdash s'$ **and**
C: *Key* (*Auth-ShaKey n*) $\notin$ *spied s*
show *Key* (*Auth-ShaKey n*) $\notin$ *spied* s'
proof (*insert B C*, *auto simp add*: *rel-def*)
fix *K*
assume (*Spy*, *Crypt K* (*Key* (*ShaK* (*Auth-ShaK n*)))) $\in$ *s*
hence *Crypt K* (*Key* (*Auth-ShaKey n*)) $\in$ *parts* (*used s*) **by** *auto*

    hence $\{Key\ (Auth\text{-}ShaKey\ n),\ Key\ K\} \subseteq spied\ s$
      by (*rule parts-crypt-key* [*OF A*], *simp add*: *image-def*)
    with *C* **show** *False* **by** *simp*
  next
    fix *Y*
    assume $(Spy,\ \{\!|Key\ (ShaK\ (Auth\text{-}ShaK\ n)),\ Y|\!\}) \in s$
    hence $\{\!|Key\ (Auth\text{-}ShaKey\ n),\ Y|\!\} \in parts\ (used\ s)$ **by** *auto*
    hence $\{Key\ (Auth\text{-}ShaKey\ n),\ Y\} \subseteq spied\ s$
      by (*rule parts-mpair-key* [*OF A*, **where** $K = Auth\text{-}ShaKey\ n$], *simp*)
    with *C* **show** *False* **by** *simp*
  next
    fix *X*
    assume $(Spy,\ \{\!|X,\ Key\ (ShaK\ (Auth\text{-}ShaK\ n))|\!\}) \in s$
    hence $\{\!|X,\ Key\ (Auth\text{-}ShaKey\ n)|\!\} \in parts\ (used\ s)$ **by** *auto*
    hence $\{X,\ Key\ (Auth\text{-}ShaKey\ n)\} \subseteq spied\ s$
      by (*rule parts-mpair-key* [*OF A*, **where** $K = Auth\text{-}ShaKey\ n$],
       simp add: *image-def*)
    with *C* **show** *False* **by** *simp*
  qed
qed

theorem *auth-prikey-secret*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin bad\text{-}prikey$
  shows $Auth\text{-}PriKey\ n \notin spied\ s$
proof
  assume $Auth\text{-}PriKey\ n \in spied\ s$
  moreover have $Auth\text{-}PriKey\ n \notin spied\ s_0$
    using *B* **by** *auto*
  ultimately have $\exists\, u\ v.\ s_0 \models u \wedge u \vdash v \wedge v \models s \wedge$
    $Auth\text{-}PriKey\ n \notin spied\ u \wedge Auth\text{-}PriKey\ n \in spied\ v$
    by (*rule rtrancl-start* [*OF A*])
  then obtain *u v* **where** *C*: $s_0 \models u$ **and** *D*: $u \vdash v$ **and**
    E: $Auth\text{-}PriKey\ n \notin spied\ u$ **and** *F*: $Auth\text{-}PriKey\ n \in spied\ v$
    by *blast*
  have $\exists\, B.\ (Auth\text{-}PriK\ n \otimes B \in spied\ u \vee B \otimes Auth\text{-}PriK\ n \in spied\ u) \wedge$
    $PriKey\ B \in spied\ u$
  proof (*insert D E F*, *auto simp add*: *rel-def*, *rule-tac* [!] *FalseE*)
    assume $Key\ (PriK\ (Auth\text{-}PriK\ n)) \notin used\ u$
    moreover have $Auth\text{-}PriKey\ n \in used\ u$
      by (*rule auth-prikey-used* [*OF C*])
    ultimately show *False* **by** *simp*
  next
    fix *K*
    assume $(Spy,\ Crypt\ K\ (Key\ (PriK\ (Auth\text{-}PriK\ n)))) \in u$
    hence $Crypt\ K\ (PriKey\ (Auth\text{-}PriK\ n)) \in parts\ (used\ u)$ **by** *auto*
    hence $(\exists\, m.\ K = Auth\text{-}ShaKey\ m \wedge$

$(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (PriKey\ (Auth\text{-}PriK\ n))) \in u) \vee$
$\{PriKey\ (Auth\text{-}PriK\ n),\ Key\ K\} \subseteq spied\ u$
by (*rule parts-crypt-prikey* [*OF C*])
then obtain *m* **where**
$(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (Auth\text{-}PriKey\ n)) \in u$
using *E* **by** *auto*
thus *False*
by (*rule auth-prikey-asset-i* [*OF C*])
next
fix *Y*
assume $(Spy,\ \{\!|Key\ (PriK\ (Auth\text{-}PriK\ n)),\ Y|\!\}) \in u$
hence $\{\!|Auth\text{-}PriKey\ n,\ Y|\!\} \in parts\ (used\ u)$ **by** *auto*
hence $\{Auth\text{-}PriKey\ n,\ Y\} \subseteq spied\ u$
by (*rule parts-mpair-key* [*OF C*, **where** $K = PriK\ (Auth\text{-}PriK\ n)$], *simp*)
thus *False*
using *E* **by** *simp*
next
fix *X*
assume $(Spy,\ \{\!|X,\ Key\ (PriK\ (Auth\text{-}PriK\ n))|\!\}) \in u$
hence $\{\!|X,\ Auth\text{-}PriKey\ n|\!\} \in parts\ (used\ u)$ **by** *auto*
hence $\{X,\ Auth\text{-}PriKey\ n\} \subseteq spied\ u$
by (*rule parts-mpair-key* [*OF C*, **where** $K = PriK\ (Auth\text{-}PriK\ n)$], *simp*
add: *image-def*)
thus *False*
using *E* **by** *simp*
qed
then obtain *B* **where** *G*: $PriKey\ B \in spied\ u$ **and**
$Auth\text{-}PriK\ n \otimes B \in spied\ u \vee B \otimes Auth\text{-}PriK\ n \in spied\ u$
by *blast*
hence $Auth\text{-}PriK\ n \otimes B \in parts\ (used\ u) \vee B \otimes Auth\text{-}PriK\ n \in parts\ (used\ u)$
by *blast*
moreover have $B \otimes Auth\text{-}PriK\ n \notin parts\ (used\ u)$
proof
assume $B \otimes Auth\text{-}PriK\ n \in parts\ (used\ u)$
hence $(\exists\, m.\ B = Auth\text{-}PriK\ m \wedge$
$(Asset\ m,\ \{\!|Num\ 2,\ PubKey\ (Auth\text{-}PriK\ n)|\!\}) \in u) \vee$
$\{PriKey\ B,\ PriKey\ (Auth\text{-}PriK\ n)\} \subseteq spied\ u$
by (*rule parts-mult* [*OF C*])
then obtain *m* **where** $(Asset\ m,\ \{\!|Num\ 2,\ Auth\text{-}PubKey\ n|\!\}) \in u$
using *E* **by** *auto*
thus *False*
by (*rule auth-pubkey-asset-ii* [*OF C*])
qed
ultimately have $Auth\text{-}PriK\ n \otimes B \in parts\ (used\ u)$ **by** *simp*
hence $(\exists\, m.\ Auth\text{-}PriK\ n = Auth\text{-}PriK\ m \wedge$
$(Asset\ m,\ \{\!|Num\ 2,\ PubKey\ B|\!\}) \in u) \vee$
$\{PriKey\ (Auth\text{-}PriK\ n),\ PriKey\ B\} \subseteq spied\ u$
by (*rule parts-mult* [*OF C*])
then obtain *m* **where** $Auth\text{-}PriK\ n = Auth\text{-}PriK\ m$ **and**

$(Asset\ m,\ \{\!|Num\ 2,\ PubKey\ B|\!\}) \in u$
using E **by** *auto*
moreover from *this* **have** $Auth\text{-}PriKey\ m \in spied\ u\ \wedge$
$(\exists\, C\ SK.\ (Asset\ m,\ Token\ m\ (Auth\text{-}PriK\ m)\ B\ C\ SK) \in u)$
by (*rule-tac asset-ii-spied* [*OF C G*])
ultimately show *False*
using E **by** *simp*
qed

proposition *asset-ii-secret*:
$[\![s_0 \models s;\ n \notin bad\text{-}prikey;\ (Asset\ n,\ \{\!|Num\ 2,\ PubKey\ B|\!\}) \in s]\!] \Longrightarrow$
$PriKey\ B \notin spied\ s$
by (*rule notI*, *frule asset-ii-spied*, *assumption+*, *drule auth-prikey-secret*, *simp+*)

proposition *asset-i-secret* [*rule-format*]:
assumes
A: $s_0 \models s$ **and**
B: $n \notin bad\text{-}shakey$
shows $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \in s \longrightarrow$
$PriKey\ S \notin spied\ s$
proof (*rule rtrancl-induct* [*OF A*], *simp add*: *image-def*, *rule impI*)
fix $s\ s'$
assume
C: $s_0 \models s$ **and**
D: $s \vdash s'$ **and**
E: $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \in s \longrightarrow$
$PriKey\ S \notin spied\ s$ **and**
F: $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \in s'$
show $PriKey\ S \notin spied\ s'$
proof (*insert D E F*, *auto simp add*: *rel-def*)
assume $(Asset\ n,\ Crypt\ (ShaK\ (Auth\text{-}ShaK\ n))\ (Key\ (PriK\ S))) \in s$
hence $(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \in s$ **by** *simp*
hence $PriKey\ S \in used\ s$
by (*rule asset-i-used* [*OF C*, *THEN mp*])
moreover assume $Key\ (PriK\ S) \notin used\ s$
ultimately show *False* **by** *simp*
next
fix K
assume $(Spy,\ Crypt\ K\ (Key\ (PriK\ S))) \in s$
hence $Crypt\ K\ (PriKey\ S) \in parts\ (used\ s)$ **by** *auto*
hence $(\exists\, m.\ K = Auth\text{-}ShaKey\ m\ \wedge$
$(Asset\ m,\ Crypt\ (Auth\text{-}ShaKey\ m)\ (PriKey\ S)) \in s)\ \vee$
$\{PriKey\ S,\ Key\ K\} \subseteq spied\ s$
(**is** $(\exists\, m.\ ?P\ m \wedge ?Q\ m) \vee$ -)
by (*rule parts-crypt-prikey* [*OF C*])
moreover assume $(Spy,\ Key\ (PriK\ S)) \notin s$
ultimately obtain m **where** G: $?P\ m \wedge ?Q\ m$ **by** *auto*
hence $?Q\ m$ **..**

```
  moreover assume
    (Asset n, Crypt (ShaK (Auth-ShaK n)) (Key (PriK S))) ∈ s
  hence (Asset n, Crypt (Auth-ShaKey n) (PriKey S)) ∈ s by simp
  ultimately have m = n
    by (rule asset-i-unique [OF C])
  moreover assume (Spy, Key (InvK K)) ∈ s
  ultimately have Key (Auth-ShaKey n) ∈ spied s
    using G by simp
  moreover have Key (Auth-ShaKey n) ∉ spied s
    by (rule auth-shakey-secret [OF C B])
  ultimately show False by contradiction
next
  fix B
  assume (Spy, S ⊗ B) ∈ s
  hence S ⊗ B ∈ parts (used s) by blast
  hence (∃ m. S = Auth-PriK m ∧ (Asset m, {|Num 2, PubKey B|}) ∈ s) ∨
    {PriKey S, PriKey B} ⊆ spied s
    (is (∃ m. ?P m ∧ -) ∨ -)
    by (rule parts-mult [OF C])
  moreover assume (Spy, Key (PriK S)) ∉ s
  ultimately obtain m where ?P m by auto
  moreover assume
    (Asset n, Crypt (ShaK (Auth-ShaK n)) (Key (PriK S))) ∈ s
  ultimately have (Asset n, Crypt (Auth-ShaKey n) (Auth-PriKey m)) ∈ s
    by simp
  thus False
    by (rule auth-prikey-asset-i [OF C])
next
  fix A
  assume (Spy, A ⊗ S) ∈ s
  hence A ⊗ S ∈ parts (used s) by blast
  hence (∃ m. A = Auth-PriK m ∧ (Asset m, {|Num 2, PubKey S|}) ∈ s) ∨
    {PriKey A, PriKey S} ⊆ spied s
    (is (∃ m. - ∧ ?P m) ∨ -)
    by (rule parts-mult [OF C])
  moreover assume (Spy, Key (PriK S)) ∉ s
  ultimately obtain m where ?P m by auto
  assume (Asset n, Crypt (ShaK (Auth-ShaK n)) (Key (PriK S))) ∈ s
  hence (Asset n, Crypt (Auth-ShaKey n) (PriKey S)) ∈ s by simp
  thus False
    by (rule asset-i-asset-ii [OF C - ‹?P m›])
next
  fix Y
  assume (Spy, {|Key (PriK S), Y|}) ∈ s
  hence {|PriKey S, Y|} ∈ parts (used s) by auto
  hence {PriKey S, Y} ⊆ spied s
    by (rule parts-mpair-key [OF C, where K = PriK S], simp)
  moreover assume (Spy, Key (PriK S)) ∉ s
  ultimately show False by simp
```

next
fix X
assume $(Spy, \{\!|X, Key\ (PriK\ S)|\!\}) \in s$
hence $\{\!|X, PriKey\ S|\!\} \in parts\ (used\ s)$ **by** *auto*
hence $\{X, PriKey\ S\} \subseteq spied\ s$
by (*rule parts-mpair-key* [*OF C*, **where** $K = PriK\ S$], *simp add*: *image-def*)
moreover assume $(Spy, Key\ (PriK\ S)) \notin s$
ultimately show *False* **by** *simp*
qed
qed

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 (*erule rtrancl-induct*, *simp add*: *image-def*, *rule impI*)
fix $s\ s'$
assume
A: $s_0 \models s$ **and**
B: $s \vdash s'$ **and**
C: $(Owner\ n, \{\!|Num\ 1, PubKey\ A|\!\}) \in s \longrightarrow PriKey\ A \notin spied\ s$ **and**
D: $(Owner\ n, \{\!|Num\ 1, PubKey\ A|\!\}) \in s'$
show $PriKey\ A \notin spied\ s'$
proof (*insert B C D*, *auto simp add*: *rel-def*)
assume $(Owner\ n, \{\!|Num\ (Suc\ 0), Key\ (PubK\ A)|\!\}) \in s$
hence $(Owner\ n, \{\!|Num\ 1, PubKey\ A|\!\}) \in s$ **by** *simp*
hence $PriKey\ A \in used\ s$
by (*rule owner-ii-used* [*OF A*, *THEN mp*])
moreover assume $Key\ (PriK\ A) \notin used\ s$
ultimately show *False* **by** *simp*
next
fix K
assume $(Spy, Crypt\ K\ (Key\ (PriK\ A))) \in s$
hence $Crypt\ K\ (PriKey\ A) \in parts\ (used\ s)$ **by** *auto*
hence $(\exists\, m.\ K = Auth\text{-}ShaKey\ m\ \wedge$
$(Asset\ m, Crypt\ (Auth\text{-}ShaKey\ m)\ (PriKey\ A)) \in s) \vee$
$\{PriKey\ A, Key\ K\} \subseteq spied\ s$
(**is** $(\exists\, m.\ \text{-} \wedge\ ?P\ m) \vee \text{-}$)
by (*rule parts-crypt-prikey* [*OF A*])
moreover assume $(Spy, Key\ (PriK\ A)) \notin s$
ultimately obtain m **where** $?P\ m$ **by** *auto*
moreover assume $(Owner\ n, \{\!|Num\ (Suc\ 0), Key\ (PubK\ A)|\!\}) \in s$
hence $(Owner\ n, \{\!|Num\ 1, PubKey\ A|\!\}) \in s$ **by** *simp*
ultimately show *False*
by (*rule asset-i-owner-ii* [*OF A*])
next
fix B
assume $(Spy, A \otimes B) \in s$
hence $A \otimes B \in parts\ (used\ s)$ **by** *blast*

hence $(\exists\, m.\ A = \mathit{Auth\text{-}PriK}\ m \wedge (\mathit{Asset}\ m,\ \{\!|\mathit{Num}\ 2,\ \mathit{PubKey}\ B|\!\}) \in s) \vee$
$\{\mathit{PriKey}\ A,\ \mathit{PriKey}\ B\} \subseteq \mathit{spied}\ s$
(**is** $(\exists\, m.\ ?P\ m \wedge \text{-}) \vee \text{-}$)
by $(\mathit{rule\ parts\text{-}mult}\ [\mathit{OF}\ A])$
moreover assume $(\mathit{Spy},\ \mathit{Key}\ (\mathit{PriK}\ A)) \notin s$
ultimately obtain m **where** $?P\ m$ **by** *auto*
moreover assume $(\mathit{Owner}\ n,\ \{\!|\mathit{Num}\ (\mathit{Suc}\ 0),\ \mathit{Key}\ (\mathit{PubK}\ A)|\!\}) \in s$
ultimately have $(\mathit{Owner}\ n,\ \{\!|\mathit{Num}\ 1,\ \mathit{Auth\text{-}PubKey}\ m|\!\}) \in s$ **by** *simp*
thus *False*
by $(\mathit{rule\ auth\text{-}pubkey\text{-}owner\text{-}ii}\ [\mathit{OF}\ A])$
next
fix B
assume $(\mathit{Spy},\ B \otimes A) \in s$
hence $B \otimes A \in \mathit{parts}\ (\mathit{used}\ s)$ **by** *blast*
hence $(\exists\, m.\ B = \mathit{Auth\text{-}PriK}\ m \wedge (\mathit{Asset}\ m,\ \{\!|\mathit{Num}\ 2,\ \mathit{PubKey}\ A|\!\}) \in s) \vee$
$\{\mathit{PriKey}\ B,\ \mathit{PriKey}\ A\} \subseteq \mathit{spied}\ s$
(**is** $(\exists\, m.\ \text{-} \wedge ?P\ m) \vee \text{-}$)
by $(\mathit{rule\ parts\text{-}mult}\ [\mathit{OF}\ A])$
moreover assume $(\mathit{Spy},\ \mathit{Key}\ (\mathit{PriK}\ A)) \notin s$
ultimately obtain m **where** $?P\ m$ **by** *auto*
moreover assume $(\mathit{Owner}\ n,\ \{\!|\mathit{Num}\ (\mathit{Suc}\ 0),\ \mathit{Key}\ (\mathit{PubK}\ A)|\!\}) \in s$
hence $(\mathit{Owner}\ n,\ \{\!|\mathit{Num}\ 1,\ \mathit{PubKey}\ A|\!\}) \in s$ **by** *simp*
ultimately show *False*
by $(\mathit{rule\ asset\text{-}ii\text{-}owner\text{-}ii}\ [\mathit{OF}\ A])$
next
fix Y
assume $(\mathit{Spy},\ \{\!|\mathit{Key}\ (\mathit{PriK}\ A),\ Y|\!\}) \in s$
hence $\{\!|\mathit{PriKey}\ A,\ Y|\!\} \in \mathit{parts}\ (\mathit{used}\ s)$ **by** *auto*
hence $\{\mathit{PriKey}\ A,\ Y\} \subseteq \mathit{spied}\ s$
by (*rule parts-mpair-key* [*OF A*, **where** $K = \mathit{PriK}\ A$], *simp*)
moreover assume $(\mathit{Spy},\ \mathit{Key}\ (\mathit{PriK}\ A)) \notin s$
ultimately show *False* **by** *simp*
next
fix X
assume $(\mathit{Spy},\ \{\!|X,\ \mathit{Key}\ (\mathit{PriK}\ A)|\!\}) \in s$
hence $\{\!|X,\ \mathit{PriKey}\ A|\!\} \in \mathit{parts}\ (\mathit{used}\ s)$ **by** *auto*
hence $\{X,\ \mathit{PriKey}\ A\} \subseteq \mathit{spied}\ s$
by (*rule parts-mpair-key* [*OF A*, **where** $K = \mathit{PriK}\ A$], *simp add*: *image-def*)
moreover assume $(\mathit{Spy},\ \mathit{Key}\ (\mathit{PriK}\ A)) \notin s$
ultimately show *False* **by** *simp*
qed
qed

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

proof (*erule rtrancl-induct*, *simp add*: *image-def*, *rule impI*)
  fix s s'
  assume
    A: $s_0 \models s$ **and**
    B: $s \vdash s'$ **and**
    C: *SesKey SK* $\in$ *spied s* $\longrightarrow$ ($\exists$ *S A C*. *?P S A C s*) **and**
    D: *SesKey SK* $\in$ *spied* s'
  show $\exists$ *S A C*. *?P S A C* s'
  proof (*insert B C D*, *auto simp add*: *rel-def*, *blast*, *rule-tac* [!] *FalseE*)
    fix K
    assume (*Spy*, *Crypt K* (*Key* (*SesK SK*))) $\in$ *s*
    hence *Crypt K* (*Key* (*SesK SK*)) $\in$ *parts* (*used s*) **by** *blast*
    hence {*Key* (*SesK SK*), *Key K*} $\subseteq$ *spied s*
      by (*rule parts-crypt-key* [*OF A*], *simp add*: *image-def*)
    moreover assume (*Spy*, *Key* (*SesK SK*)) $\notin$ *s*
    ultimately show *False* **by** *simp*
  next
    fix Y
    assume (*Spy*, $\{\!|$*Key* (*SesK SK*), $Y|\!\}$) $\in$ *s*
    hence $\{\!|$*SesKey SK*, $Y|\!\}$ $\in$ *parts* (*used s*) **by** *auto*
    hence {*SesKey SK*, Y} $\subseteq$ *spied s*
      by (*rule parts-mpair-key* [*OF A*, **where** K = *SesK SK*], *simp*)
    moreover assume (*Spy*, *Key* (*SesK SK*)) $\notin$ *s*
    ultimately show *False* **by** *simp*
  next
    fix X
    assume (*Spy*, $\{\!|X$, *Key* (*SesK SK*)$|\!\}$) $\in$ *s*
    hence $\{\!|X$, *SesKey SK*$|\!\}$ $\in$ *parts* (*used s*) **by** *auto*
    hence {X, *SesKey SK*} $\subseteq$ *spied s*
      by (*rule parts-mpair-key* [*OF A*, **where** K = *SesK SK*], *simp add*: *image-def*)
    moreover assume (*Spy*, *Key* (*SesK SK*)) $\notin$ *s*
    ultimately show *False* **by** *simp*
  qed
qed

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
  have ($\exists$ *S*. *fst SK* = *Some S* $\wedge$ *Crypt* (*Auth-ShaKey n*) (*PriKey S*) $\in$ *used s*) $\vee$
    fst SK = *None*
    (**is** ($\exists$ *S*. *?P S*) $\vee$ -)
    by (*rule owner-seskey-nonce* [*OF A C*])
  moreover assume *SesKey SK* $\in$ *spied s*
  hence D: $\exists$ *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*

by (*rule seskey-spied* [*OF A*])
ultimately obtain S **where** *?P S* **by** *auto*
hence *Crypt* (*Auth-ShaKey n*) (*PriKey S*) $\in$ *parts* (*used s*) **by** *blast*
hence ($\exists$ *m*. *Auth-ShaKey n* = *Auth-ShaKey m* $\wedge$
(*Asset m*, *Crypt* (*Auth-ShaKey m*) (*PriKey S*)) $\in$ *s*) $\vee$
{*PriKey S*, *Key* (*Auth-ShaKey n*)} $\subseteq$ *spied s*
(**is** ($\exists$ *m*. *?Q m* $\wedge$ *?R m*) $\vee$ -)
by (*rule parts-crypt-prikey* [*OF A*])
moreover have *Key* (*Auth-ShaKey n*) $\notin$ *spied s*
by (*rule auth-shakey-secret* [*OF A B*])
ultimately obtain *m* **where** *?Q m* **and** *?R m* **by** *blast*
hence *m* $\notin$ *bad-shakey*
using *B* **by** *simp*
hence *PriKey S* $\notin$ *spied s*
by (*rule asset-i-secret* [*OF A* - ‹*?R m*›])
moreover have *PriKey S* $\in$ *spied s*
using *D* **and** ‹*?P S*› **by** *auto*
ultimately show *False* **by** *contradiction*
qed

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
have $\exists$ *A B C D*. *fst* (*snd SK*) = {*A*, *B*} $\wedge$ *snd* (*snd SK*) = {*C*, *D*} $\wedge$
(*Owner m*, {|*Num 1*, *PubKey A*|}) $\in$ *s* $\wedge$
(*Owner m*, {|*Num 3*, *PubKey C*|}) $\in$ *s* $\wedge$
(*Owner m*, *Crypt* (*SesK SK*) (*PubKey D*)) $\in$ *s*
(**is** $\exists$ *A B C D*. *?P A B* $\wedge$ - $\wedge$ *?Q A* $\wedge$ -)
by (*rule owner-seskey-other* [*OF A C*])
then obtain *A B'* **where** *?P A B'* **and** *?Q A* **by** *blast*
assume *SesKey SK* $\in$ *spied s*
hence $\exists$ *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** $\exists$ *S A' C*. - $\wedge$ *?R A'* $\wedge$ -)
by (*rule seskey-spied* [*OF A*])
then obtain *A'* **where** *A'* $\in$ *fst* (*snd SK*) **and** *PriKey A'* $\in$ *spied s* (**is** *?R A'*)
by *blast*
hence {*A'*, *A*, *B*} $\subseteq$ {*A*, *B'*}
using *E* **and** ‹*?P A B'*› **by** *simp*
hence *card* {*A'*, *A*, *B*} $\leq$ *card* {*A*, *B'*}
by (*rule-tac card-mono*, *simp*)
also have ... $\leq$ *Suc* (*Suc 0*)
by (*rule card-insert-le-m1*, *simp-all*)

```
  finally have card {A′, A, B} ≤ Suc (Suc 0) .
  moreover have card {A′, A, B} = Suc (card {A, B})
  proof (rule card-insert-disjoint, simp-all, rule conjI, rule-tac [!] notI)
    assume A′ = A
    hence ?R A
      using ‹?R A′› by simp
    moreover have ¬ ?R A
      by (rule owner-ii-secret [OF A ‹?Q A›])
    ultimately show False by contradiction
  next
    assume A′ = B
    hence ?R B
      using ‹?R A′› by simp
    moreover have ¬ ?R B
      by (rule asset-ii-secret [OF A B D])
    ultimately show False by contradiction
  qed
  moreover have card {A, B} = Suc (card {B})
  proof (rule card-insert-disjoint, simp-all, rule notI)
    assume A = B
    hence (Asset n, {|Num 2, PubKey A|}) ∈ s
      using D by simp
    thus False
      by (rule asset-ii-owner-ii [OF A - ‹?Q A›])
  qed
  ultimately show False by simp
qed

proposition asset-seskey-shakey:
  assumes
    A: s0 |= s and
    B: n ∉ bad-shakey and
    C: (Asset n, SesKey SK) ∈ s
  shows SesKey SK ∉ spied s
proof
  have ∃ S. fst SK = Some S ∧
    (Asset n, Crypt (Auth-ShaKey n) (PriKey S)) ∈ s
    (is ∃ S. ?P S ∧ ?Q S)
    by (rule asset-seskey-nonce [OF A C])
  then obtain S where ?P S and ?Q S by blast
  have PriKey S ∉ spied s
    by (rule asset-i-secret [OF A B ‹?Q S›])
  moreover assume SesKey SK ∈ spied s
  hence ∃ S A C. fst SK = Some S ∧ A ∈ fst (snd SK) ∧ C ∈ snd (snd SK) ∧
    {PriKey S, PriKey A, PriKey C} ⊆ spied s
    by (rule seskey-spied [OF A])
  hence PriKey S ∈ spied s
    using ‹?P S› by auto
  ultimately show False by contradiction
```

qed

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 $m = n$
proof (*rule ccontr*)
  have *D*: (*Owner m*, *SesKey SK*) $\in s \wedge$
    ($\exists$ *A B C*. *Token m A B C SK* $\in$ *used s* $\wedge$ *B* $\in$ *fst* (*snd SK*))
    (**is** *?P m* $\wedge$ ($\exists$ *A B C*. *?Q m A B C*))
    by (*rule owner-v-state* [*OF A B*])
  hence *?P m* **by** *blast*
  hence $\exists$ *A B C D*. *fst* (*snd SK*) = {*A*, *B*} $\wedge$ *snd* (*snd SK*) = {*C*, *D*} $\wedge$
    (*Owner m*, ⦃*Num 1*, *PubKey A*⦄) $\in s \wedge$
    (*Owner m*, ⦃*Num 3*, *PubKey C*⦄) $\in s \wedge$
    (*Owner m*, *Crypt* (*SesK SK*) (*PubKey D*)) $\in$ *s*
    (**is** $\exists$ *A B C D*. *?R A B* $\wedge$ *?S C D* $\wedge$ *?T m A* $\wedge$ *?U m C D*)
    by (*rule owner-seskey-other* [*OF A*])
  then obtain *A B* **where** *?R A B* **and** *?T m A* **by** *blast*
  have *?P n* $\wedge$ ($\exists$ *A B C*. *?Q n A B C*)
    by (*rule owner-v-state* [*OF A C*])
  hence *?P n* **by** *blast*
  hence $\exists$ *A B C D*. *?R A B* $\wedge$ *?S C D* $\wedge$ *?T n A* $\wedge$ *?U n C D*
    by (*rule owner-seskey-other* [*OF A*])
  then obtain A' B' **where** *?R* A' B' **and** *?T n* A' **by** *blast*
  from *D* **obtain** A'' B'' *C* **where** *?Q m* A'' B'' *C* **by** *blast*
  hence *Token m* A'' B'' *C SK* $\in$ *parts* (*used s*) **by** *blast*
  hence *Crypt* (*SesK SK*) ($A'' \otimes B''$) $\in$ *parts* (*used s*) **by** *blast*
  hence $B'' \in$ *fst* (*snd SK*) $\wedge$
    ($\exists$ *i* C'. (*Asset i*, *Token i* (*Auth-PriK i*) B'' C' *SK*) $\in$ *s*) $\vee$
    {$A'' \otimes B''$, *SesKey SK*} $\subseteq$ *spied s*
    (**is** *?V* B'' $\wedge$ ($\exists$ *i* C'. *?W i* B'' C') $\vee$ -)
    by (*rule parts-crypt-mult* [*OF A*])
  hence $\exists$ *D*. *?V D* $\wedge$ $D \notin \{A, A'\}$
  proof (*rule disjE*, (*erule-tac conjE*, ((*erule-tac exE*)+)?)+)
    fix *i* C'
    assume *?V* B'' **and** *?W i* B'' C'
    have $\exists$ *A D*. *?R A* B'' $\wedge$ *?S* C' *D* $\wedge$
      (*Asset i*, ⦃*Num 2*, *PubKey* B''⦄) $\in s \wedge$ (*Asset i*, ⦃*Num 4*, *PubKey D*⦄) $\in s \wedge$
      Crypt (*SesK SK*) (*PubKey D*) $\in$ *used s* $\wedge$ (*Asset i*, *PubKey* B'') $\in$ *s*
      (**is** $\exists$ *A D*. - $\wedge$ - $\wedge$ *?X i* B'' $\wedge$ -)
      by (*rule asset-iv-state* [*OF A* ‹*?W i* B'' C'›])
    hence *?X i* B'' **by** *blast*
    have $B'' \neq A$
    proof
      assume $B'' = A$

```
    hence ?X i A
      using ‹?X i B''› by simp
    thus False
      by (rule asset-ii-owner-ii [OF A - ‹?T m A›])
  qed
  moreover have B'' ≠ A'
  proof
    assume B'' = A'
    hence ?X i A'
      using ‹?X i B''› by simp
    thus False
      by (rule asset-ii-owner-ii [OF A - ‹?T n A'›])
  qed
  ultimately show ?thesis
    using ‹?V B''› by blast
next
  assume {A'' ⊗ B'', SesKey SK} ⊆ spied s
  hence SesKey SK ∈ spied s by simp
  hence ∃ S D E. fst SK = Some S ∧ ?V D ∧ E ∈ snd (snd SK) ∧
    {PriKey S, PriKey D, PriKey E} ⊆ spied s
    by (rule seskey-spied [OF A])
  then obtain D where ?V D and PriKey D ∈ spied s (is ?X D)
    by blast
  moreover have D ≠ A
  proof
    assume D = A
    hence ?X A
      using ‹?X D› by simp
    moreover have ¬ ?X A
      by (rule owner-ii-secret [OF A ‹?T m A›])
    ultimately show False by contradiction
  qed
  moreover have D ≠ A'
  proof
    assume D = A'
    hence ?X A'
      using ‹?X D› by simp
    moreover have ¬ ?X A'
      by (rule owner-ii-secret [OF A ‹?T n A'›])
    ultimately show False by contradiction
  qed
  ultimately show ?thesis by blast
qed
then obtain D where ?V D and E: D ∉ {A, A'} by blast
hence {D, A, A'} ⊆ {A, B}
  using ‹?R A B› and ‹?R A' B'› by blast
hence card {D, A, A'} ≤ card {A, B}
  by (rule-tac card-mono, simp)
also have ... ≤ Suc (Suc 0)
```

by (*rule card-insert-le-m1*, *simp-all*)
finally have *card* $\{D, A, A'\} \leq$ *Suc* (*Suc 0*) **.**
moreover have *card* $\{D, A, A'\} =$ *Suc* (*card* $\{A, A'\}$)
by (*rule card-insert-disjoint* [*OF - E*], *simp*)
moreover assume $m \neq n$
hence *card* $\{A, A'\} =$ *Suc* (*card* $\{A'\}$)
proof (*rule-tac card-insert-disjoint*, *simp-all*, *erule-tac contrapos-nn*)
assume $A = A'$
hence *?T n A*
using ‹*?T n A'*› **by** *simp*
thus $m = n$
by (*rule owner-ii-unique* [*OF A* ‹*?T m A*›])
qed
ultimately show *False* **by** *simp*
qed

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*
proof –
have (*Owner n*, *SesKey SK*) $\in s \land$
($\exists$ *A B C*. *Token n A B C SK* $\in$ *used s* $\land$ *B* $\in$ *fst* (*snd SK*))
(**is** *?P* $\land$ ($\exists$ *A B C*. *?Q A B C* $\land$ *?R B*))
by (*rule owner-v-state* [*OF A C*])
then obtain *A B C* **where** *?P* **and** *?Q A B C* **and** *?R B* **by** *blast*
have $n \in$ *bad-shakey* $\lor$ $n \notin$ *bad-shakey* **by** *simp*
moreover {
assume $n \in$ *bad-shakey*
hence *D*: $n \notin$ *bad-prikey*
using *B* **by** *simp*
hence *Auth-PriKey n* $\notin$ *spied s*
by (*rule auth-prikey-secret* [*OF A*])
moreover have *Sign n A* $\in$ *parts* (*used s*)
using ‹*?Q A B C*› **by** *blast*
hence *A* = *Auth-PriK n*
by (*rule parts-sign* [*OF A*])
hence *?Q* (*Auth-PriK n*) *B C*
using ‹*?Q A B C*› **by** *simp*
hence *Auth-PriK n* $\otimes$ *B* $\in$ *parts* (*used s*) **by** *blast*
hence ($\exists$ *m*. *Auth-PriK n* = *Auth-PriK m* $\land$
(*Asset m*, $\{\!|$*Num 2*, *PubKey B*$|\!\}$) $\in s$) $\lor$
$\{$*PriKey* (*Auth-PriK n*), *PriKey B*$\} \subseteq$ *spied s*
(**is** ($\exists$ *m*. *?S m* $\land$ *?T m*) $\lor$ -)
by (*rule parts-mult* [*OF A*])
ultimately obtain *m* **where** *?S m* **and** *?T m* **by** *auto*

```
    hence m ∉ bad-prikey
      using D by simp
    hence ?thesis
      by (rule owner-seskey-prikey [OF A - ‹?P› ‹?T m› ‹?R B›])
  }
  moreover {
    assume n ∉ bad-shakey
    hence ?thesis
      by (rule owner-seskey-shakey [OF A - ‹?P›])
  }
  ultimately show ?thesis ..
qed


theorem owner-num-genuine:
  assumes
    A: s₀ ⊨ s and
    B: n ∉ bad-shakey ∩ bad-prikey and
    C: (Owner n, Crypt (SesK SK) (Pwd n)) ∈ s and
    D: Crypt (SesK SK) (Num 0) ∈ used s
  shows (Asset n, Crypt (SesK SK) (Num 0)) ∈ s
proof −
  have Crypt (SesK SK) (Num 0) ∈ parts (used s)
    using D ..
  hence (∃ m. (Asset m, Crypt (SesK SK) (Num 0)) ∈ s) ∨
    SesKey SK ∈ spied s
    by (rule parts-crypt-num [OF A])
  moreover have E: SesKey SK ∉ spied s
    by (rule owner-seskey-secret [OF A B C])
  ultimately obtain m where (Asset m, Crypt (SesK SK) (Num 0)) ∈ s
    by blast
  moreover from this have (Asset m, SesKey SK) ∈ s ∧
    Crypt (SesK SK) (Pwd m) ∈ used s
    by (rule asset-v-state [OF A])
  hence Crypt (SesK SK) (Pwd m) ∈ parts (used s) by blast
  hence (∃ SK'. SesK SK = SesK SK' ∧
    (Owner m, Crypt (SesK SK') (Pwd m)) ∈ s) ∨
    {Pwd m, Key (SesK SK)} ⊆ spied s
    by (rule parts-crypt-pwd [OF A])
  hence (Owner m, Crypt (SesK SK) (Pwd m)) ∈ s
    using E by simp
  hence m = n
    by (rule owner-seskey-unique [OF A - C])
  ultimately show ?thesis by simp
qed


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** $?P\ n\ A \wedge ?Q\ B \wedge ?R\ C \wedge ?S\ n\ B \wedge$ -)
proof –
have $Crypt\ (SesK\ SK)\ (Sign\ n\ A) \in parts\ (used\ s)$
using E **by** *blast*
hence $(Asset\ n,\ SesKey\ SK) \in s \vee SesKey\ SK \in spied\ s$
by (*rule parts-crypt-sign* [*OF A*])
moreover have $SesKey\ SK \notin spied\ s$
by (*rule owner-seskey-secret* [*OF A B D*])
ultimately have $(Asset\ n,\ SesKey\ SK) \in s$ **by** *simp*
hence $\exists\, A\ B\ C\ D.\ fst\ (snd\ SK) = \{A,\ B\} \wedge snd\ (snd\ SK) = \{C,\ D\} \wedge$
$?S\ n\ B \wedge (Asset\ n,\ \{\!|Num\ 4,\ PubKey\ D|\!\}) \in s \wedge$
$(Asset\ n,\ Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK) \in s$
(**is** $\exists\, A\ B\ C\ D.\ ?T\ A\ B \wedge ?U\ C\ D \wedge$ - $\wedge\ ?V\ n\ D \wedge ?W\ n\ B\ C$)
by (*rule asset-seskey-other* [*OF A*])
then obtain $A'\ B'\ C'\ D$ **where**
$?T\ A'\ B'$ **and** $?U\ C'\ D$ **and** $?S\ n\ B'$ **and** $?V\ n\ D$ **and** $?W\ n\ B'\ C'$
by *blast*
have $Sign\ n\ A \in parts\ (used\ s)$
using E **by** *blast*
hence $?P\ n\ A$
by (*rule parts-sign* [*OF A*])
have $Crypt\ (SesK\ SK)\ (A \otimes B) \in parts\ (used\ s)$
using E **by** *blast*
hence $?Q\ B \wedge (\exists\, m\ C''.\ ?W\ m\ B\ C'') \vee \{A \otimes B,\ SesKey\ SK\} \subseteq spied\ s$
by (*rule parts-crypt-mult* [*OF A*])
moreover have F: $SesKey\ SK \notin spied\ s$
by (*rule owner-seskey-secret* [*OF A B D*])
ultimately obtain $m\ C''$ **where** $?Q\ B$ **and** $?W\ m\ B\ C''$ **by** *blast*
have $\exists\, A\ D.\ ?T\ A\ B \wedge ?U\ C''\ D \wedge ?S\ m\ B \wedge ?V\ m\ D \wedge$
$Crypt\ (SesK\ SK)\ (PubKey\ D) \in used\ s \wedge (Asset\ m,\ PubKey\ B) \in s$
by (*rule asset-iv-state* [*OF A* ‹$?W\ m\ B\ C''$›])
hence $?S\ m\ B$ **by** *blast*
have $(Owner\ n,\ SesKey\ SK) \in s \wedge$
$(\exists\, A\ B\ C.\ Token\ n\ A\ B\ C\ SK \in used\ s \wedge B \in fst\ (snd\ SK))$
by (*rule owner-v-state* [*OF A D*])
hence $(Owner\ n,\ SesKey\ SK) \in s$ **by** *blast*
hence $\exists\, A\ B\ C\ D.\ ?T\ A\ B \wedge ?U\ C\ D \wedge$
$(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\}) \in s \wedge$
$(Owner\ n,\ \{\!|Num\ 3,\ PubKey\ C|\!\}) \in s \wedge$
$(Owner\ n,\ Crypt\ (SesK\ SK)\ (PubKey\ D)) \in s$
(**is** $\exists\, A\ B\ C\ D.$ - $\wedge$ - $\wedge\ ?X\ A \wedge$ -)
by (*rule owner-seskey-other* [*OF A*])

then obtain A'' **where** *?Q* A'' **and** *?X* A'' **by** *blast*
have G: $B' = B$
proof (*rule ccontr*)
  have $\{A'', B', B\} \subseteq \{A', B'\}$
    using ‹*?T* A' B'› **and** ‹*?Q* B› **and** ‹*?Q* A''› **by** *simp*
  hence *card* $\{A'', B', B\} \leq$ *card* $\{A', B'\}$
    by (*rule-tac card-mono*, *simp*)
  also have $\ldots \leq$ *Suc* (*Suc 0*)
    by (*rule card-insert-le-m1*, *simp-all*)
  finally have *card* $\{A'', B', B\} \leq$ *Suc* (*Suc 0*) **.**
  moreover have $A'' \notin \{B', B\}$
  proof (*simp*, *rule conjI*, *rule-tac* [!] *notI*)
    assume $A'' = B'$
    hence *?S n* A''
      using ‹*?S n* B'› **by** *simp*
    thus *False*
      by (*rule asset-ii-owner-ii* [*OF A* - ‹*?X* A''›])
  next
    assume $A'' = B$
    hence *?S m* A''
      using ‹*?S m B*› **by** *simp*
    thus *False*
      by (*rule asset-ii-owner-ii* [*OF A* - ‹*?X* A''›])
  qed
  hence *card* $\{A'', B', B\}$ = *Suc* (*card* $\{B', B\}$)
    by (*rule-tac card-insert-disjoint*, *simp*)
  moreover assume $B' \neq B$
  hence *card* $\{B', B\}$ = *Suc* (*card* $\{B\}$)
    by (*rule-tac card-insert-disjoint*, *simp-all*)
  ultimately show *False* **by** *simp*
qed
hence *?S n B*
  using ‹*?S n* B'› **by** *simp*
have *Crypt* (*SesK SK*) (*PubKey C*) $\in$ *parts* (*used s*)
  using *E* **by** *blast*
hence *?R C* $\wedge$ (($\exists$ *n*. (*Owner n*, *SesKey SK*) $\in$ *s*) $\vee$ ($\exists$ *n B*. *?W n B C*)) $\vee$
  SesKey SK $\in$ *spied s*
  by (*rule parts-crypt-pubkey* [*OF A*])
hence *?R C*
  using *F* **by** *simp*
hence $C \in \{C', D\}$
  using ‹*?U* C' *D*› **by** *simp*
moreover have $C \neq D$
proof
  assume $C = D$
  hence *?V n C*
    using ‹*?V n D*› **by** *simp*
  thus *False*
    by (*rule asset-iii-owner-iii* [*OF A* - *C*])

qed
ultimately have $C = C'$ **by** *simp*
hence (*Asset n*, *Token n A B C SK*) $\in$ *s*
using *G* **and** ‹*?P n A*› **and** ‹*?W n B' C'*› **by** *simp*
thus *?thesis*
using ‹*?P n A*› **and** ‹*?Q B*› **and** ‹*?R C*› **and** ‹*?S n B*› **by** *simp*
qed

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*
proof (*rule rtrancl-induct* [*OF A*], *insert B*, *simp add*: *image-def*)
fix *s s'*
assume
C: $s_0 \models s$ **and**
D: $s \vdash s'$ **and**
E: *Pwd n* $\notin$ *spied s*
show *Pwd n* $\notin$ *spied s'*
proof (*insert D E*, *auto simp add*: *rel-def*)
fix *K*
assume (*Spy*, *Crypt K* (*Pwd n*)) $\in$ *s*
hence *Crypt K* (*Pwd n*) $\in$ *parts* (*used s*) **by** *blast*
hence ($\exists$ *SK*. *K* = *SesK SK* $\wedge$ (*Owner n*, *Crypt* (*SesK SK*) (*Pwd n*)) $\in$ *s*) $\vee$
{*Pwd n*, *Key K*} $\subseteq$ *spied s*
(**is** ($\exists$ *SK*. *?P SK* $\wedge$ *?Q SK*) $\vee$ -)
by (*rule parts-crypt-pwd* [*OF C*])
then obtain *SK* **where** *?P SK* **and** *?Q SK*
using *E* **by** *blast*
have *n* $\notin$ *bad-shakey* $\cap$ *bad-prikey*
using *B* **by** *simp*
hence *SesKey SK* $\notin$ *spied s*
by (*rule owner-seskey-secret* [*OF C* - ‹*?Q SK*›])
moreover assume (*Spy*, *Key* (*InvK K*)) $\in$ *s*
ultimately show *False*
using ‹*?P SK*› **by** *simp*
next
fix *Y*
assume (*Spy*, ⦃*Pwd n*, *Y*⦄) $\in$ *s*
hence ⦃*Pwd n*, *Y*⦄ $\in$ *parts* (*used s*) **by** *blast*
hence {*Pwd n*, *Y*} $\subseteq$ *spied s*
by (*rule parts-mpair-pwd* [*OF C*, **where** *n* = *n*], *simp*)
with *E* **show** *False* **by** *simp*
next
fix *X*
assume (*Spy*, ⦃*X*, *Pwd n*⦄) $\in$ *s*
hence ⦃*X*, *Pwd n*⦄ $\in$ *parts* (*used s*) **by** *blast*

hence $\{X,\ \mathit{Pwd}\ n\} \subseteq \mathit{spied}\ s$
by $(\mathit{rule\ parts\text{-}mpair\text{-}pwd}\ [\mathit{OF}\ C,$ **where** $n = n],\ \mathit{simp})$
with E **show** *False* **by** *simp*
qed
qed

theorem *asset-seskey-unique*:
assumes
A: $s_0 \models s$ **and**
B: $(\mathit{Asset}\ m,\ \mathit{Token}\ m\ (\mathit{Auth\text{-}PriK}\ m)\ B'\ C'\ \mathit{SK}) \in s$ **and**
C: $(\mathit{Asset}\ n,\ \mathit{Token}\ n\ (\mathit{Auth\text{-}PriK}\ n)\ B\ C\ \mathit{SK}) \in s$
(**is** $?P\ n\ B\ C\ \mathit{SK}\ s$)
shows $m = n \wedge B' = B \wedge C' = C$
proof $(\mathit{subst}\ (2)\ \mathit{cases\text{-}simp}\ [\mathit{of}\ B' = B,\ \mathit{symmetric}],\ \mathit{simp},\ \mathit{rule\ conjI},\ \mathit{rule\ impI},$
$\mathit{insert}\ B\ C,\ \mathit{simp\ only}{:},\ \mathit{drule\ asset\text{-}iv\text{-}unique}\ [\mathit{OF}\ A],\ \mathit{simp},\ \mathit{simp},\ \mathit{rule\ ccontr})$
assume $B' \neq B$
moreover have $\exists A\ D.\ \mathit{fst}\ (\mathit{snd}\ \mathit{SK}) = \{A,\ B'\} \wedge \mathit{snd}\ (\mathit{snd}\ \mathit{SK}) = \{C',\ D\} \wedge$
$(\mathit{Asset}\ m,\ \{\!|\mathit{Num}\ 2,\ \mathit{PubKey}\ B'|\!\}) \in s \wedge (\mathit{Asset}\ m,\ \{\!|\mathit{Num}\ 4,\ \mathit{PubKey}\ D|\!\}) \in s \wedge$
$\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{PubKey}\ D) \in \mathit{used}\ s \wedge (\mathit{Asset}\ m,\ \mathit{PubKey}\ B') \in s$
(**is** $?Q\ m\ B'\ C'$)
by $(\mathit{rule\ asset\text{-}iv\text{-}state}\ [\mathit{OF}\ A\ B])$
then obtain A **where** $\mathit{fst}\ (\mathit{snd}\ \mathit{SK}) = \{A,\ B'\}$ **and**
$(\mathit{Asset}\ m,\ \{\!|\mathit{Num}\ 2,\ \mathit{PubKey}\ B'|\!\}) \in s$
by *blast*
moreover have $?Q\ n\ B\ C$
by $(\mathit{rule\ asset\text{-}iv\text{-}state}\ [\mathit{OF}\ A\ C])$
hence $B \in \mathit{fst}\ (\mathit{snd}\ \mathit{SK})$ **and** $(\mathit{Asset}\ n,\ \{\!|\mathit{Num}\ 2,\ \mathit{PubKey}\ B|\!\}) \in s$
by *auto*
ultimately have D: $\forall A \in \mathit{fst}\ (\mathit{snd}\ \mathit{SK}).$
$\exists i\ C.\ (\mathit{Asset}\ i,\ \{\!|\mathit{Num}\ 2,\ \mathit{PubKey}\ A|\!\}) \in s \wedge\ ?P\ i\ A\ C\ \mathit{SK}\ s$
using B **and** C **by** *auto*
have $\mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{PubKey}\ C) \in \mathit{parts}\ (\mathit{used}\ s)$
using C **by** *blast*
thus *False*
proof $(\mathit{rule\ parts\text{-}pubkey\text{-}false}\ [\mathit{OF}\ A],\ \mathit{rule\text{-}tac\ allI},\ \mathit{rule\text{-}tac}\ [!]\ \mathit{notI})$
fix i
assume $(\mathit{Owner}\ i,\ \mathit{SesKey}\ \mathit{SK}) \in s$
hence $\exists A\ B\ C\ D.\ \mathit{fst}\ (\mathit{snd}\ \mathit{SK}) = \{A,\ B\} \wedge \mathit{snd}\ (\mathit{snd}\ \mathit{SK}) = \{C,\ D\} \wedge$
$(\mathit{Owner}\ i,\ \{\!|\mathit{Num}\ 1,\ \mathit{PubKey}\ A|\!\}) \in s \wedge$
$(\mathit{Owner}\ i,\ \{\!|\mathit{Num}\ 3,\ \mathit{PubKey}\ C|\!\}) \in s \wedge$
$(\mathit{Owner}\ i,\ \mathit{Crypt}\ (\mathit{SesK}\ \mathit{SK})\ (\mathit{PubKey}\ D)) \in s$
by $(\mathit{rule\ owner\text{-}seskey\text{-}other}\ [\mathit{OF}\ A])$
then obtain A **where** $A \in \mathit{fst}\ (\mathit{snd}\ \mathit{SK})$ **and**
E: $(\mathit{Owner}\ i,\ \{\!|\mathit{Num}\ 1,\ \mathit{PubKey}\ A|\!\}) \in s$
by *blast*
then obtain j **where** $(\mathit{Asset}\ j,\ \{\!|\mathit{Num}\ 2,\ \mathit{PubKey}\ A|\!\}) \in s$
using D **by** *blast*
thus *False*

by (*rule asset-ii-owner-ii* [*OF A - E*])
next
assume *SesKey SK* $\in$ *spied s*
hence $\exists$ *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** *?R s*)
by (*rule seskey-spied* [*OF A*])
moreover have $\neg$ ($\exists$ *A* $\in$ *fst* (*snd SK*). *PriKey A* $\in$ *spied s*)
(**is** $\neg$ *?S s*)
proof
assume *?S s*
moreover have $\neg$ *?S* s_0
by (*subst bex-simps*, *rule ballI*, *drule bspec* [*OF D*], (*erule exE*)+,
erule conjE, *rule asset-ii-init* [*OF A*])
ultimately have $\exists$ *u v*. $s_0 \models u \wedge u \vdash v \wedge v \models s \wedge \neg$ *?S u* $\wedge$ *?S v*
by (*rule rtrancl-start* [*OF A*])
then obtain *u v A* **where** *E*: $s_0 \models u$ **and** *F*: $u \vdash v$ **and** *G*: $v \models s$ **and**
H: $\neg$ *?S u* **and** *I*: *A* $\in$ *fst* (*snd SK*) **and** *J*: *PriKey A* $\notin$ *spied u* **and**
K: *PriKey A* $\in$ *spied v*
by *blast*
then obtain *i* **where** (*Asset i*, {|*Num 2*, *PubKey A*|}) $\in$ *s*
using *D* **by** *blast*
hence (*Asset i*, {|*Num 2*, *PubKey A*|}) $\in$ *v*
proof (*rule-tac ccontr*, *drule-tac rtrancl-start* [*OF G*], *simp*,
(*erule-tac exE*)+, (*erule-tac conjE*)+)
fix *w x*
assume $w \vdash x$ **and** (*Asset i*, {|*Num 2*, *PubKey A*|}) $\notin$ *w* **and**
(*Asset i*, {|*Num 2*, *PubKey A*|}) $\in$ *x*
hence *PriKey A* $\notin$ *spied w*
by (*auto simp add*: *rel-def*)
moreover assume $v \models w$
hence *PriKey A* $\in$ *spied w*
by (*rule-tac rev-subsetD* [*OF K*], *rule spied-subset*)
ultimately show *False* **by** *contradiction*
qed
hence (*Asset i*, {|*Num 2*, *PubKey A*|}) $\in$ *u*
using *F* **and** *K* **by** (*auto simp add*: *rel-def*)
hence *Auth-PriKey i* $\in$ *spied u* $\wedge$ ($\exists$ *C SK*. *?P i A C SK u*)
by (*rule asset-ii-spied-start* [*OF E F K J*])
then obtain *C' SK'* **where** *L*: *?P i A C' SK' u* **by** *blast*
moreover have *M*: $u \models s$
using *F* **and** *G* **by** *simp*
ultimately have *?P i A C' SK' s*
by (*erule-tac rev-subsetD*, *rule-tac state-subset*)
moreover obtain *j C* **where** *?P j A C SK s*
using *D* **and** *I* **by** *blast*
ultimately have $i = j \wedge C' = C \wedge SK' = SK$
by (*rule asset-iv-unique* [*OF A*])
hence *Crypt* (*SesK SK*) (*PubKey C*) $\in$ *parts* (*used u*)

using *L* **by** *blast*
thus *False*
proof (*rule parts-pubkey-false* [*OF E*], *rule-tac allI*, *rule-tac* [!] *notI*)
fix *i*
assume (*Owner i*, *SesKey SK*) ∈ *u*
hence ∃ *A B C D*. *fst* (*snd SK*) = {*A*, *B*} ∧ *snd* (*snd SK*) = {*C*, *D*} ∧
(*Owner i*, ⦃*Num 1*, *PubKey A*⦄) ∈ *u* ∧
(*Owner i*, ⦃*Num 3*, *PubKey C*⦄) ∈ *u* ∧
(*Owner i*, *Crypt* (*SesK SK*) (*PubKey D*)) ∈ *u*
by (*rule owner-seskey-other* [*OF E*])
then obtain *A* **where** *A* ∈ *fst* (*snd SK*) **and**
N: (*Owner i*, ⦃*Num 1*, *PubKey A*⦄) ∈ *u*
by *blast*
then obtain *j* **where** (*Asset j*, ⦃*Num 2*, *PubKey A*⦄) ∈ *s*
using *D* **by** *blast*
moreover have (*Owner i*, ⦃*Num 1*, *PubKey A*⦄) ∈ *s*
by (*rule rev-subsetD* [*OF N*], *rule state-subset* [*OF M*])
ultimately show *False*
by (*rule asset-ii-owner-ii* [*OF A*])
next
assume *SesKey SK* ∈ *spied u*
hence *?R u*
by (*rule seskey-spied* [*OF E*])
thus *False*
using *H* **by** *blast*
qed
qed
ultimately show *False* **by** *blast*
qed
qed

theorem *asset-seskey-secret*:
assumes
A: s_0 ⊨ *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 −
have *D*: (*Asset n*, *SesKey SK*) ∈ *s* ∧ *Crypt* (*SesK SK*) (*Pwd n*) ∈ *used s*
by (*rule asset-v-state* [*OF A C*])
have *n* ∈ *bad-shakey* ∨ *n* ∉ *bad-shakey* **by** *simp*
moreover {
assume *n* ∈ *bad-shakey*
hence *Pwd n* ∉ *spied s*
using *B* **by** (*rule-tac pwd-secret* [*OF A*], *simp*)
moreover have *Crypt* (*SesK SK*) (*Pwd n*) ∈ *parts* (*used s*)
using *D* **by** *blast*
hence (∃ *SK'*. *SesK SK* = *SesK SK'* ∧

$(Owner\ n,\ Crypt\ (SesK\ SK')\ (Pwd\ n)) \in s) \vee$
$\{Pwd\ n,\ Key\ (SesK\ SK)\} \subseteq spied\ s$
by (*rule parts-crypt-pwd* [*OF A*])
ultimately have $(Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s$ **by** *simp*
hence *?thesis*
using *B* **by** (*rule-tac owner-seskey-secret* [*OF A*], *simp-all*)
}
moreover {
assume $n \notin bad\text{-}shakey$
hence *?thesis*
using *D* **by** (*rule-tac asset-seskey-shakey* [*OF A*], *simp-all*)
}
ultimately show *?thesis* **..**
qed

theorem *asset-pwd-genuine*:
assumes
A: $s_0 \models s$ **and**
B: $n \notin bad\text{-}shakey \cap (bad\text{-}pwd \cup bad\text{-}prikey)$ **and**
C: $(Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0)) \in s$
shows $(Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s$
proof −
have $(Asset\ n,\ SesKey\ SK) \in s \wedge Crypt\ (SesK\ SK)\ (Pwd\ n) \in used\ s$
by (*rule asset-v-state* [*OF A C*])
hence $Crypt\ (SesK\ SK)\ (Pwd\ n) \in parts\ (used\ s)$ **by** *blast*
hence $(\exists\, SK'.\ SesK\ SK = SesK\ SK' \wedge$
$(Owner\ n,\ Crypt\ (SesK\ SK')\ (Pwd\ n)) \in s) \vee$
$\{Pwd\ n,\ Key\ (SesK\ SK)\} \subseteq spied\ s$
by (*rule parts-crypt-pwd* [*OF A*])
moreover have $SesKey\ SK \notin spied\ s$
by (*rule asset-seskey-secret* [*OF A B C*])
ultimately show *?thesis* **by** *simp*
qed

theorem *asset-token-genuine*:
assumes
A: $s_0 \models s$ **and**
B: $n \notin bad\text{-}shakey \cap (bad\text{-}pwd \cup bad\text{-}prikey)$ **and**
C: $(Asset\ n,\ \{\!|Num\ 4,\ PubKey\ D|\!\}) \in s$ **and**
D: $(Asset\ n,\ Crypt\ (SesK\ SK)\ (Num\ 0)) \in s$ **and**
E: $D \in snd\ (snd\ SK)$
shows $(Owner\ n,\ Crypt\ (SesK\ SK)\ (PubKey\ D)) \in s$
proof −
have $(Owner\ n,\ Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s$
by (*rule asset-pwd-genuine* [*OF A B D*])
hence $(Owner\ n,\ SesKey\ SK) \in s \wedge$
$(\exists\, A\ B\ C.\ Token\ n\ A\ B\ C\ SK \in used\ s \wedge B \in fst\ (snd\ SK))$

by (*rule owner-v-state* [*OF A*])
hence (*Owner n*, *SesKey SK*) $\in$ *s* **..**
hence $\exists$ *A B C D*. *fst* (*snd SK*) = {*A*, *B*} $\wedge$ *snd* (*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*
(**is** $\exists$ *A B C D*. - $\wedge$ *?P C D* $\wedge$ - $\wedge$ *?Q C* $\wedge$ *?R D*)
by (*rule owner-seskey-other* [*OF A*])
then obtain *C D'* **where** *?P C D'* **and** *?Q C* **and** *?R D'* **by** *blast*
have $D \neq C$
proof
assume $D = C$
hence *?Q D*
using ‹*?Q C*› **by** *simp*
thus *False*
by (*rule asset-iii-owner-iii* [*OF A C*])
qed
hence $D = D'$
using *E* **and** ‹*?P C D'*› **by** *simp*
thus *?thesis*
using ‹*?R D'*› **by** *simp*
qed

end

3 Anonymity properties

theory *Anonymity*
imports *Authentication*
begin

proposition *crypts-empty* [*simp*]:
crypts {} = {}
by (*rule equalityI*, *rule subsetI*, *erule crypts.induct*, *simp-all*)

proposition *crypts-mono*:
$H \subseteq H' \Longrightarrow$ *crypts* $H \subseteq$ *crypts* H'
by (*rule subsetI*, *erule crypts.induct*, *auto*)

lemma *crypts-union-1*:
crypts $(H \cup H') \subseteq$ *crypts* $H \cup$ *crypts* H'
by (*rule subsetI*, *erule crypts.induct*, *auto*)

lemma *crypts-union-2*:
crypts $H \cup$ *crypts* $H' \subseteq$ *crypts* $(H \cup H')$
by (*rule subsetI*, *erule UnE*, *erule-tac* [!] *crypts.induct*, *auto*)

proposition *crypts-union*:

crypts $(H \cup H') = $ *crypts* $H \cup$ *crypts* H'
by (*rule equalityI*, *rule crypts-union-1*, *rule crypts-union-2*)

proposition *crypts-insert*:
crypts (*insert* X H) = *crypts-msg* $X \cup$ *crypts* H
by (*simp only*: *insert-def crypts-union*, *subst crypts-msg-def*, *simp*)

proposition *crypts-msg-num* [*simp*]:
crypts-msg (*Num* n) = $\{$*Num* $n\}$
by (*subst crypts-msg-def*, *rule equalityI*, *rule subsetI*, *erule crypts.induct*, *simp*, *rotate-tac* [*1–3*], *erule-tac* [*1–3*] *rev-mp*, *rule-tac* [*1–3*] *list.induct*, *simp-all*, *blast*)

proposition *crypts-msg-agent* [*simp*]:
crypts-msg (*Agent* n) = $\{$*Agent* $n\}$
by (*subst crypts-msg-def*, *rule equalityI*, *rule subsetI*, *erule crypts.induct*, *simp*, *rotate-tac* [*1–3*], *erule-tac* [*1–3*] *rev-mp*, *rule-tac* [*1–3*] *list.induct*, *simp-all*, *blast*)

proposition *crypts-msg-pwd* [*simp*]:
crypts-msg (*Pwd* n) = $\{$*Pwd* $n\}$
by (*subst crypts-msg-def*, *rule equalityI*, *rule subsetI*, *erule crypts.induct*, *simp*, *rotate-tac* [*1–3*], *erule-tac* [*1–3*] *rev-mp*, *rule-tac* [*1–3*] *list.induct*, *simp-all*, *blast*)

proposition *crypts-msg-key* [*simp*]:
crypts-msg (*Key* K) = $\{$*Key* $K\}$
by (*subst crypts-msg-def*, *rule equalityI*, *rule subsetI*, *erule crypts.induct*, *simp*, *rotate-tac* [*1–3*], *erule-tac* [*1–3*] *rev-mp*, *rule-tac* [*1–3*] *list.induct*, *simp-all*, *blast*)

proposition *crypts-msg-mult* [*simp*]:
crypts-msg $(A \otimes B) = \{A \otimes B\}$
by (*subst crypts-msg-def*, *rule equalityI*, *rule subsetI*, *erule crypts.induct*, *simp*, *rotate-tac* [*1–3*], *erule-tac* [*1–3*] *rev-mp*, *rule-tac* [*1–3*] *list.induct*, *simp-all*, *blast*)

lemma *crypts-hash-1*:
crypts $\{$*Hash* $X\} \subseteq$ *insert* (*Hash* X) (*crypts* $\{X\}$)
by (*rule subsetI*, *erule crypts.induct*, *simp-all*, (*erule disjE*, *rotate-tac*, *erule rev-mp*, *rule list.induct*, *simp-all*, *blast*, (*drule crypts-hash*, *simp*)*?*)+)

lemma *crypts-hash-2*:
insert (*Hash* X) (*crypts* $\{X\}$) $\subseteq$ *crypts* $\{$*Hash* $X\}$
by (*rule subsetI*, *simp*, *erule disjE*, *blast*, *erule crypts.induct*, *simp*, *subst id-apply* [*symmetric*], *subst foldr-Nil* [*symmetric*], *rule crypts-hash*, *simp*, *blast*+)

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

crypts-msg (*Hash X*) = *insert* (*Hash X*) (*crypts-msg X*)
by (*simp add*: *crypts-msg-def*, *rule equalityI*, *rule crypts-hash-1*, *rule crypts-hash-2*)

proposition *crypts-comp*:
$X \in$ *crypts H* $\Longrightarrow$ *Crypt K X* $\in$ *crypts* (*Crypt K* ‘ *H*)
by (*erule crypts.induct*, *blast*, (*simp only*: *comp-apply*
[*symmetric*, **where** f = *Crypt K*] *foldr-Cons* [*symmetric*],
(*erule crypts-hash* | *erule crypts-fst* | *erule crypts-snd*))+)

lemma *crypts-crypt-1*:
crypts {*Crypt K X*} $\subseteq$ *Crypt K* ‘ *crypts* {*X*}
by (*rule subsetI*, *erule crypts.induct*, *fastforce*, *rotate-tac* [!], *erule-tac* [!] *rev-mp*,
rule-tac [!] *list.induct*, *auto*)

lemma *crypts-crypt-2*:
Crypt K ‘ *crypts* {*X*} $\subseteq$ *crypts* {*Crypt K X*}
by (*rule subsetI*, *simp add*: *image-iff*, *erule bexE*, *drule crypts-comp*, *simp*)

proposition *crypts-msg-crypt* [*simp*]:
crypts-msg (*Crypt K X*) = *Crypt K* ‘ *crypts-msg X*
by (*simp add*: *crypts-msg-def*, *rule equalityI*, *rule crypts-crypt-1*, *rule crypts-crypt-2*)

lemma *crypts-mpair-1*:
crypts {⦃*X*, *Y*⦄} $\subseteq$ *insert* ⦃*X*, *Y*⦄ (*crypts* {*X*} $\cup$ *crypts* {*Y*})
by (*rule subsetI*, *erule crypts.induct*, *simp-all*, (*erule disjE*, *rotate-tac*, *erule rev-mp*,
rule list.induct, (*simp*+, *blast*)+)+)

lemma *crypts-mpair-2*:
insert ⦃*X*, *Y*⦄ (*crypts* {*X*} $\cup$ *crypts* {*Y*}) $\subseteq$ *crypts* {⦃*X*, *Y*⦄}
by (*rule subsetI*, *simp*, *erule disjE*, *blast*, *erule disjE*, (*erule crypts.induct*, *simp*,
subst id-apply [*symmetric*], *subst foldr-Nil* [*symmetric*], (*rule crypts-fst* [*of - X*] |
rule crypts-snd), *rule crypts-used*, *simp*, *blast*+)+)

proposition *crypts-msg-mpair* [*simp*]:
crypts-msg ⦃*X*, *Y*⦄ = *insert* ⦃*X*, *Y*⦄ (*crypts-msg X* $\cup$ *crypts-msg Y*)
by (*simp add*: *crypts-msg-def*, *rule equalityI*, *rule crypts-mpair-1*, *rule crypts-mpair-2*)

proposition *foldr-crypt-size*:
size (*foldr Crypt KS X*) = *size X* + *length KS*
by (*induction KS*, *simp-all*)

proposition *key-sets-empty* [*simp*]:
key-sets X {} = {}
by (*simp add*: *key-sets-def*)

proposition *key-sets-mono*:
$H \subseteq H'$ $\Longrightarrow$ *key-sets X H* $\subseteq$ *key-sets X* H'
by (*auto simp add*: *key-sets-def*)

proposition *key-sets-union*:
$key\text{-}sets\ X\ (H \cup H') = key\text{-}sets\ X\ H \cup key\text{-}sets\ X\ H'$
by (*auto simp add*: *key-sets-def*)

proposition *key-sets-insert*:
$key\text{-}sets\ X\ (insert\ Y\ H) = key\text{-}sets\text{-}msg\ X\ Y \cup key\text{-}sets\ X\ H$
by (*simp only*: *insert-def key-sets-union*, *subst key-sets-msg-def*, *simp*)

proposition *key-sets-msg-eq*:
$key\text{-}sets\text{-}msg\ X\ X = \{\{\}\}$
by (*simp add*: *key-sets-msg-def key-sets-def*, *rule equalityI*, *rule subsetI*, *simp*,
erule exE, *erule rev-mp*, *rule list.induct*, *simp*, *rule impI*, *erule conjE*,
drule arg-cong [*of - X size*], *simp-all add*: *foldr-crypt-size*)

proposition *key-sets-msg-num* [*simp*]:
$key\text{-}sets\text{-}msg\ X\ (Num\ n) = (if\ X = Num\ n\ then\ \{\{\}\}\ else\ \{\})$
by (*simp add*: *key-sets-msg-eq*, *simp add*: *key-sets-msg-def key-sets-def*, *rule impI*,
rule allI, *rule list.induct*, *simp-all*)

proposition *key-sets-msg-agent* [*simp*]:
$key\text{-}sets\text{-}msg\ X\ (Agent\ n) = (if\ X = Agent\ n\ then\ \{\{\}\}\ else\ \{\})$
by (*simp add*: *key-sets-msg-eq*, *simp add*: *key-sets-msg-def key-sets-def*, *rule impI*,
rule allI, *rule list.induct*, *simp-all*)

proposition *key-sets-msg-pwd* [*simp*]:
$key\text{-}sets\text{-}msg\ X\ (Pwd\ n) = (if\ X = Pwd\ n\ then\ \{\{\}\}\ else\ \{\})$
by (*simp add*: *key-sets-msg-eq*, *simp add*: *key-sets-msg-def key-sets-def*, *rule impI*,
rule allI, *rule list.induct*, *simp-all*)

proposition *key-sets-msg-key* [*simp*]:
$key\text{-}sets\text{-}msg\ X\ (Key\ K) = (if\ X = Key\ K\ then\ \{\{\}\}\ else\ \{\})$
by (*simp add*: *key-sets-msg-eq*, *simp add*: *key-sets-msg-def key-sets-def*, *rule impI*,
rule allI, *rule list.induct*, *simp-all*)

proposition *key-sets-msg-mult* [*simp*]:
$key\text{-}sets\text{-}msg\ X\ (A \otimes B) = (if\ X = A \otimes B\ then\ \{\{\}\}\ else\ \{\})$
by (*simp add*: *key-sets-msg-eq*, *simp add*: *key-sets-msg-def key-sets-def*, *rule impI*,
rule allI, *rule list.induct*, *simp-all*)

proposition *key-sets-msg-hash* [*simp*]:
$key\text{-}sets\text{-}msg\ X\ (Hash\ Y) = (if\ X = Hash\ Y\ then\ \{\{\}\}\ else\ \{\})$
by (*simp add*: *key-sets-msg-eq*, *simp add*: *key-sets-msg-def key-sets-def*, *rule impI*,
rule allI, *rule list.induct*, *simp-all*)

lemma *key-sets-crypt-1*:
$X \neq Crypt\ K\ Y \Longrightarrow$
$key\text{-}sets\ X\ \{Crypt\ K\ Y\} \subseteq insert\ (InvKey\ K)\ `\ key\text{-}sets\ X\ \{Y\}$
by (*rule subsetI*, *simp add*: *key-sets-def*, *erule exE*, *rotate-tac*, *erule rev-mp*,

rule list.induct, *auto*)

lemma *key-sets-crypt-2*:
insert (*InvKey K*) ‘ *key-sets X* {*Y*} $\subseteq$ *key-sets X* {*Crypt K Y*}
by (*rule subsetI*, *simp add*: *key-sets-def image-iff*, (*erule exE*, *erule conjE*)+,
drule arg-cong [**where** f = *Crypt K*], *simp only*: *comp-apply*
[*symmetric*, *of Crypt K*] *foldr-Cons* [*symmetric*], *subst conj-commute*,
rule exI, *rule conjI*, *assumption*, *simp*)

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*)
by (*simp add*: *key-sets-msg-eq*, *simp add*: *key-sets-msg-def*, *rule impI*,
rule equalityI, *erule key-sets-crypt-1* [*simplified*],
rule key-sets-crypt-2 [*simplified*])

proposition *key-sets-msg-mpair* [*simp*]:
key-sets-msg X ⦃*Y*, *Z*⦄ = (*if X* = ⦃*Y*, *Z*⦄ *then* {{}} *else* {})
by (*simp add*: *key-sets-msg-eq*, *simp add*: *key-sets-msg-def key-sets-def*, *rule impI*,
rule allI, *rule list.induct*, *simp-all*)

proposition *key-sets-range*:
$U \in$ *key-sets X H* $\Longrightarrow$ $U \subseteq$ *range Key*
by (*simp add*: *key-sets-def*, *blast*)

proposition *key-sets-crypts-hash*:
key-sets (*Hash X*) (*crypts H*) $\subseteq$ *key-sets X* (*crypts H*)
by (*simp add*: *key-sets-def*, *blast*)

proposition *key-sets-crypts-fst*:
key-sets ⦃*X*, *Y*⦄ (*crypts H*) $\subseteq$ *key-sets X* (*crypts H*)
by (*simp add*: *key-sets-def*, *blast*)

proposition *key-sets-crypts-snd*:
key-sets ⦃*X*, *Y*⦄ (*crypts H*) $\subseteq$ *key-sets Y* (*crypts H*)
by (*simp add*: *key-sets-def*, *blast*)

lemma *log-spied-1*:
$[\![s \vdash s'$;
Log X $\in$ *parts* (*used s*) $\longrightarrow$ *Log X* $\in$ *spied s*;
Log X $\in$ *parts* (*used s'*)$]\!]$ $\Longrightarrow$
Log X $\in$ *spied s'*
by (*simp add*: *rel-def*, ((*erule disjE*)?, ((*erule exE*)+)?, *simp add*: *parts-insert*,
((*subst* (*asm*) *disj-assoc* [*symmetric*])?, *erule disjE*, (*drule parts-dec* |
drule parts-enc | *drule parts-sep* | *drule parts-con*), *simp*+)?)+)

proposition *log-spied* [*rule-format*]:
$s_0 \models s \Longrightarrow$

$Log\ X \in parts\ (used\ s) \longrightarrow$
$Log\ X \in spied\ s$
by (*erule rtrancl-induct*, *subst parts-init*, *simp add*: *Range-iff image-def*, *rule impI*, *rule log-spied-1*)

proposition *log-dec*:
$[\![s_0 \models s;\ s' = insert\ (Spy,\ X)\ s \wedge (Spy,\ Crypt\ K\ X) \in s \wedge$
$(Spy,\ Key\ (InvK\ K)) \in s]\!] \Longrightarrow$
$key\text{-}sets\ Y\ (crypts\ \{Y.\ Log\ Y = X\}) \subseteq key\text{-}sets\ Y\ (crypts\ (Log\ -`\ spied\ s))$
by (*rule key-sets-mono*, *rule crypts-mono*, *rule subsetI*, *simp*, *drule parts-dec* [**where** $Y = X$], *drule-tac* [!] *sym*, *simp-all*, *rule log-spied* [*simplified*])

proposition *log-sep*:
$[\![s_0 \models s;\ s' = insert\ (Spy,\ X)\ (insert\ (Spy,\ Y)\ s) \wedge (Spy,\ \{\!|X,\ Y|\!\}) \in s]\!] \Longrightarrow$
$key\text{-}sets\ Z\ (crypts\ \{Z.\ Log\ Z = X\}) \subseteq key\text{-}sets\ Z\ (crypts\ (Log\ -`\ spied\ s)) \wedge$
$key\text{-}sets\ Z\ (crypts\ \{Z.\ Log\ Z = Y\}) \subseteq key\text{-}sets\ Z\ (crypts\ (Log\ -`\ spied\ s))$
by (*rule conjI*, (*rule key-sets-mono*, *rule crypts-mono*, *rule subsetI*, *simp*, *frule parts-sep* [**where** $Z = X$], *drule-tac* [*2*] *parts-sep* [**where** $Z = Y$], *simp-all add*: *parts-msg-def*, *blast+*, *drule sym*, *simp*, *rule log-spied* [*simplified*], *assumption+*)+)

lemma *idinfo-spied-1*:
$[\![s \vdash s';$
$\langle n,\ X\rangle \in parts\ (used\ s) \longrightarrow \langle n,\ X\rangle \in spied\ s;$
$\langle n,\ X\rangle \in parts\ (used\ s')]\!] \Longrightarrow$
$\langle n,\ X\rangle \in spied\ s'$
by (*simp add*: *rel-def*, (*erule disjE*, (*erule exE*)+, *simp add*: *parts-insert*, ((*subst* (*asm*) *disj-assoc* [*symmetric*])?, *erule disjE*, (*drule parts-dec* | *drule parts-enc* | *drule parts-sep* | *drule parts-con*), *simp*+)?)+, *auto simp add*: *parts-insert*)

proposition *idinfo-spied* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\langle n,\ X\rangle \in parts\ (used\ s) \longrightarrow$
$\langle n,\ X\rangle \in spied\ s$
by (*erule rtrancl-induct*, *subst parts-init*, *simp add*: *Range-iff image-def*, *rule impI*, *rule idinfo-spied-1*)

proposition *idinfo-dec*:
$[\![s_0 \models s;\ s' = insert\ (Spy,\ X)\ s \wedge (Spy,\ Crypt\ K\ X) \in s \wedge$
$(Spy,\ Key\ (InvK\ K)) \in s;\ \langle n,\ Y\rangle = X]\!] \Longrightarrow$
$\langle n,\ Y\rangle \in spied\ s$
by (*drule parts-dec* [**where** $Y = \langle n,\ Y\rangle$], *drule sym*, *simp*, *rule idinfo-spied*)

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*
by (*drule parts-sep* [**where** $Z = \langle n, Z\rangle$], *erule disjE*, (*drule sym*, *simp*)+,
rule idinfo-spied)

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'*
by (*simp add*: *rel-def*, (*erule disjE*, (*erule exE*)+, *simp*, ((*subst* (*asm*) *disj-assoc*
[*symmetric*])?, *erule disjE*, (*drule idinfo-dec* [*OF A*] | *drule idinfo-sep* [*OF A*]),
simp+ | *erule disjE*, (*simp add*: *image-iff*)+, *blast?*)?)+)

proposition *idinfo-msg* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\langle n, X\rangle \in$ *spied s* $\longrightarrow$
$X \in$ *spied s*
by (*erule rtrancl-induct*, *simp*, *blast*, *rule impI*, *rule idinfo-msg-1*)

proposition *parts-agent*:
$[\![s_0 \models s; n \notin$ *bad-agent*$]\!] \Longrightarrow$ *Agent n* $\notin$ *parts* (*used s*)
apply (*erule rtrancl-induct*)
subgoal
apply (*subst parts-init*)
apply (*simp add*: *Range-iff image-def*)
done
subgoal
apply (*simp add*: *rel-def*)
apply ((*erule disjE*)?, (*erule exE*)+, *simp add*: *parts-insert image-iff*,
(*rule ccontr*, (*drule parts-dec* | *drule parts-enc* | *drule parts-sep* |
drule parts-con), *simp*+)?)+
done
done

lemma *idinfo-init-1* [*rule-format*]:
assumes *A*: $s_0 \models s$
shows $[\![s \vdash s'; n \notin$ *bad-id-password* $\cup$ *bad-id-pubkey* $\cup$ *bad-id-shakey*;
$\forall X.\ \langle n, X\rangle \notin$ *spied s*$]\!] \Longrightarrow$
$\langle n, X\rangle \notin$ *spied s'*
by (*rule notI*, *simp add*: *rel-def*, ((*erule disjE*)?, (*erule exE*)+, (*blast* | *simp*,
((*drule idinfo-dec* [*OF A*] | *drule idinfo-sep* [*OF A*]), *simp*, *blast* |
(*erule conjE*)+, *drule parts-agent* [*OF A*], *blast*))?)+)

proposition *idinfo-init*:
$[\![s_0 \models s; n \notin$ *bad-id-password* $\cup$ *bad-id-pubkey* $\cup$ *bad-id-shakey*$]\!] \Longrightarrow$
$\langle n, X\rangle \notin$ *spied s*
by (*induction arbitrary*: *X rule*: *rtrancl-induct*, *simp add*: *image-def*, *blast*,

rule idinfo-init-1)

lemma *idinfo-mpair-1* [*rule-format*]:
$[\![(s,\ s') \in$ *rel-id-hash* $\cup$ *rel-id-crypt* $\cup$ *rel-id-sep* $\cup$ *rel-id-con*;
$\forall X\ Y.\ \langle n,\ \{\!|X,\ Y|\!\}\rangle \in$ *spied s* $\longrightarrow$
key-sets $\{\!|X,\ Y|\!\}$ (*crypts* (*Log* $-$' *spied s*)) $\neq \{\}$;
$\langle n,\ \{\!|X,\ Y|\!\}\rangle \in$ *spied s'*$]\!] \Longrightarrow$
key-sets $\{\!|X,\ Y|\!\}$ (*crypts* (*Log* $-$' *spied s'*)) $\neq \{\}$
by ((*erule disjE*)?, *clarify?*, *simp add*: *image-iff Image-def*, (*drule subsetD*
[*OF key-sets-crypts-hash*] | *drule key-sets-range*, *blast* | (*drule spec*)+,
drule mp, *simp*, *simp add*: *ex-in-conv* [*symmetric*], *erule exE*, *frule subsetD*
[*OF key-sets-crypts-fst*], *drule subsetD* [*OF key-sets-crypts-snd*])?)+

lemma *idinfo-mpair-2* [*rule-format*]:
assumes *A*: $s_0 \models s$
shows $[\![s \vdash s';\ (s,\ s') \notin$ *rel-id-hash* $\cup$ *rel-id-crypt* $\cup$ *rel-id-sep* $\cup$ *rel-id-con*;
$\forall X\ Y.\ \langle n,\ \{\!|X,\ Y|\!\}\rangle \in$ *spied s* $\longrightarrow$
key-sets $\{\!|X,\ Y|\!\}$ (*crypts* (*Log* $-$' *spied s*)) $\neq \{\}$;
$\langle n,\ \{\!|X,\ Y|\!\}\rangle \in$ *spied s'*$]\!] \Longrightarrow$
key-sets $\{\!|X,\ Y|\!\}$ (*crypts* (*Log* $-$' *spied s'*)) $\neq \{\}$
by (*simp only*: *rel-def Un-iff de-Morgan-disj*, *simp*, ((*erule disjE*)?, (*erule exE*)+,
simp add: *Image-def*, (*simp only*: *Collect-disj-eq crypts-union key-sets-union*,
simp)?,
((*subst* (*asm*) *disj-assoc* [*symmetric*])?, *erule disjE*, (*drule idinfo-dec* [*OF A*] |
drule idinfo-sep [*OF A*]), *simp*+)?)+)

proposition *idinfo-mpair* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\langle n,\ \{\!|X,\ Y|\!\}\rangle \in$ *spied s* $\longrightarrow$
key-sets $\{\!|X,\ Y|\!\}$ (*crypts* (*Log* $-$' *spied s*)) $\neq \{\}$
proof (*induction arbitrary*: *X Y rule*: *rtrancl-induct*, *simp add*: *image-def*,
rule impI)
fix *s s' X Y*
assume
$s_0 \models s$ **and**
$s \vdash s'$ **and**
$\bigwedge X\ Y.\ \langle n,\ \{\!|X,\ Y|\!\}\rangle \in$ *spied s* $\longrightarrow$
key-sets $\{\!|X,\ Y|\!\}$ (*crypts* (*Log* $-$' *spied s*)) $\neq \{\}$ **and**
$\langle n,\ \{\!|X,\ Y|\!\}\rangle \in$ *spied s'*
thus *key-sets* $\{\!|X,\ Y|\!\}$ (*crypts* (*Log* $-$' *spied s'*)) $\neq \{\}$
by (*cases* $(s,\ s') \in$ *rel-id-hash* $\cup$ *rel-id-crypt* $\cup$ *rel-id-sep* $\cup$ *rel-id-con*,
erule-tac [*2*] *idinfo-mpair-2*, *erule-tac idinfo-mpair-1*, *simp-all*)
qed

proposition *key-sets-pwd-empty*:
$s_0 \models s \Longrightarrow$
key-sets (*Hash* (*Pwd n*)) (*crypts* (*Log* $-$' *spied s*)) $= \{\} \wedge$

key-sets ⦃*Pwd n*, *X*⦄ (*crypts* (*Log* −' *spied s*)) = {} ∧
key-sets ⦃*X*, *Pwd n*⦄ (*crypts* (*Log* −' *spied s*)) = {}
(**is** - ⟹ *key-sets ?X* (*?H s*) = - ∧ *key-sets ?Y* - = - ∧ *key-sets ?Z* - = -)
proof (*erule rtrancl-induct*, *simp add*: *image-iff Image-def*)
fix *s s'*
assume
A: s_0 ⊨ *s* **and**
B: *s* ⊢ *s'* **and**
C: *key-sets* (*Hash* (*Pwd n*)) (*?H s*) = {} ∧
key-sets ⦃*Pwd n*, *X*⦄ (*?H s*) = {} ∧
key-sets ⦃*X*, *Pwd n*⦄ (*?H s*) = {}
show *key-sets* (*Hash* (*Pwd n*)) (*?H s'*) = {} ∧
key-sets ⦃*Pwd n*, *X*⦄ (*?H s'*) = {} ∧
key-sets ⦃*X*, *Pwd n*⦄ (*?H s'*) = {}
by (*insert B C*, *simp add*: *rel-def*, ((*erule disjE*)*?*, ((*erule exE*)+)*?*, *simp add*: *image-iff Image-def*, (*simp only*: *Collect-disj-eq crypts-union key-sets-union*, *simp add*: *crypts-insert key-sets-insert*)*?*, (*frule log-dec* [*OF A*, **where** *Y* = *?X*], *frule log-dec* [*OF A*, **where** *Y* = *?Y*], *drule log-dec* [*OF A*, **where** *Y* = *?Z*] | *frule log-sep* [*OF A*, **where** *Z* = *?X*], *frule log-sep* [*OF A*, **where** *Z* = *?Y*], *drule log-sep* [*OF A*, **where** *Z* = *?Z*])*?*)+)
qed

proposition *key-sets-pwd-seskey* [*rule-format*]:
s_0 ⊨ *s* ⟹
U ∈ *key-sets* (*Pwd n*) (*crypts* (*Log* −' *spied s*)) ⟶
(∃ *SK*. *U* = {*SesKey SK*} ∧
((*Owner n*, *Crypt* (*SesK SK*) (*Pwd n*)) ∈ *s* ∨
(*Asset n*, *Crypt* (*SesK SK*) (*Num 0*)) ∈ *s*))
(**is** - ⟹ - ⟶ *?P s*)
proof (*erule rtrancl-induct*, *simp add*: *image-iff Image-def*, *rule impI*)
fix *s s'*
assume
A: s_0 ⊨ *s* **and**
B: *s* ⊢ *s'* **and**
C: *U* ∈ *key-sets* (*Pwd n*) (*crypts* (*Log* −' *spied s*)) ⟶ *?P s* **and**
D: *U* ∈ *key-sets* (*Pwd n*) (*crypts* (*Log* −' *spied s'*))
show *?P s'*
by (*insert B C D*, *simp add*: *rel-def*, ((*erule disjE*)*?*, ((*erule exE*)+)*?*, *simp add*: *image-iff Image-def*, (*simp only*: *Collect-disj-eq crypts-union key-sets-union*, *simp add*: *crypts-insert key-sets-insert split*: *if-split-asm*, *blast?*)*?*, (*erule disjE*, (*drule log-dec* [*OF A*] | *drule log-sep* [*OF A*]), (*erule conjE*)*?*, *drule subsetD*, *simp*)*?*)+)
qed

lemma *pwd-anonymous-1* [*rule-format*]:
⟦s_0 ⊨ *s*; *n* ∉ *bad-id-password*⟧ ⟹
⟨*n*, *Pwd n*⟩ ∈ *spied s* ⟶

($\exists$ *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*)
proof (*erule rtrancl-induct*, *simp add*: *image-def*, *rule impI*)
fix $s\ s'$
assume
A: $s_0 \models s$ **and**
B: $s \vdash s'$ **and**
C: $\langle n,\ Pwd\ n\rangle \in spied\ s \longrightarrow\ ?P\ s$ **and**
D: $\langle n,\ Pwd\ n\rangle \in spied\ s'$
show *?P* s'
by (*insert B C D*, *simp add*: *rel-def*, ((*erule disjE*)*?*, (*erule exE*)+, *simp add*: *image-iff*, *blast?*, ((*subst* (*asm*) *disj-assoc* [*symmetric*])*?*, *erule disjE*, (*drule idinfo-dec* [*OF A*] | *drule idinfo-sep* [*OF A*]), *simp*, *blast+* | *insert key-sets-pwd-empty* [*OF A*], *clarsimp*)*?*, (((*erule disjE*)*?*, *erule conjE*, *drule sym*, *simp*, (*drule key-sets-pwd-seskey* [*OF A*] | *drule idinfo-mpair* [*OF A*, *simplified*]), *simp*)+ | *drule key-sets-range*, *blast*)*?*)+)
qed

theorem *pwd-anonymous*:
assumes
A: $s_0 \models s$ **and**
B: $n \notin$ *bad-id-password* **and**
C: $n \notin$ *bad-shakey* $\cap$ (*bad-pwd* $\cup$ *bad-prikey*) $\cap$ (*bad-id-pubkey* $\cup$ *bad-id-shak*)
shows $\langle n,\ Pwd\ n\rangle \notin spied\ s$
proof
assume *D*: $\langle n,\ Pwd\ n\rangle \in spied\ s$
hence $n \in$ *bad-id-password* $\cup$ *bad-id-pubkey* $\cup$ *bad-id-shakey*
by (*rule contrapos-pp*, *rule-tac idinfo-init* [*OF A*])
moreover have $\exists$ *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** $\exists$ *SK*. *?P SK* $\wedge$ (*?Q SK* $\vee$ *?R SK*))
by (*rule pwd-anonymous-1* [*OF A B D*])
then obtain *SK* **where** *?P SK* **and** *?Q SK* $\vee$ *?R SK* **by** *blast*
moreover {
assume *?Q SK*
hence $n \in$ *bad-shakey* $\cap$ *bad-prikey*
by (*rule-tac contrapos-pp* [*OF* ‹*?P SK*›], *rule-tac owner-seskey-secret* [*OF A*])
}
moreover {
assume *?R SK*
hence $n \in$ *bad-shakey* $\cap$ (*bad-pwd* $\cup$ *bad-prikey*)
by (*rule-tac contrapos-pp* [*OF* ‹*?P SK*›], *rule-tac asset-seskey-secret* [*OF A*])
}
ultimately show *False*
using *B* **and** *C* **by** *blast*
qed

proposition *idinfo-pwd-start*:
  assumes
    A: $s_0 \models s$ **and**
    B: $n \notin$ *bad-agent*
  shows $[\![s \vdash s'; \exists X.\ \langle n,\ X\rangle \in \mathit{spied}\ s' \wedge X \neq \mathit{Pwd}\ n;$
    $\neg\ (\exists X.\ \langle n,\ X\rangle \in \mathit{spied}\ s \wedge X \neq \mathit{Pwd}\ n)]\!] \Longrightarrow$
      $\exists SK.\ \mathit{SesKey}\ SK \in \mathit{spied}\ s \wedge$
        $((\mathit{Owner}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ SK)\ (\mathit{Pwd}\ n)) \in s \vee$
         $(\mathit{Asset}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ SK)\ (\mathit{Num}\ 0)) \in s)$
proof (*simp add*: *rel-def*, *insert parts-agent* [*OF A B*], *insert key-sets-pwd-empty* [*OF A*], (*erule disjE*, (*erule exE*)+, *simp*, *erule conjE*, (*subst* (*asm*) *disj-assoc* [*symmetric*])?, (*erule disjE*)?, (*drule idinfo-dec* [*OF A*] | *drule idinfo-sep* [*OF A*] | *drule spec*, *drule mp*), *simp*+)+, *auto*, *rule FalseE*, *rule-tac* [*3*] *FalseE*)
  fix *X U K*
  assume $\forall X.\ (\mathit{Spy},\ \langle n,\ X\rangle) \in s \longrightarrow X = \mathit{Pwd}\ n$ **and** $(\mathit{Spy},\ \langle n,\ K\rangle) \in s$
  hence $K = \mathit{Pwd}\ n$ **by** *simp*
  moreover assume $U \in \mathit{key\text{-}sets}\ X\ (\mathit{crypts}\ (\mathit{Log}\ -`\ \mathit{spied}\ s))$
  hence $U \subseteq \mathit{range}\ \mathit{Key}$
    by (*rule key-sets-range*)
  moreover assume $K \in U$
  ultimately show *False* **by** *blast*
next
  fix *X U*
  assume $\forall X.\ (\mathit{Spy},\ \langle n,\ X\rangle) \in s \longrightarrow X = \mathit{Pwd}\ n$ **and** $(\mathit{Spy},\ \langle n,\ X\rangle) \in s$
  hence *C*: $X = \mathit{Pwd}\ n$ **by** *simp*
  moreover assume $U \in \mathit{key\text{-}sets}\ X\ (\mathit{crypts}\ (\mathit{Log}\ -`\ \mathit{spied}\ s))$
  ultimately have $U \in \mathit{key\text{-}sets}\ (\mathit{Pwd}\ n)\ (\mathit{crypts}\ (\mathit{Log}\ -`\ \mathit{spied}\ s))$ **by** *simp*
  hence $\exists SK.\ U = \{\mathit{SesKey}\ SK\} \wedge$
    $((\mathit{Owner}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ SK)\ (\mathit{Pwd}\ n)) \in s \vee$
     $(\mathit{Asset}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ SK)\ (\mathit{Num}\ 0)) \in s)$
    by (*rule key-sets-pwd-seskey* [*OF A*])
  moreover assume $U \subseteq \mathit{spied}\ s$
  ultimately show $\exists x\ U\ V.\ (\mathit{Spy},\ \mathit{Key}\ (\mathit{SesK}\ (x,\ U,\ V))) \in s \wedge$
    $((\mathit{Owner}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ (x,\ U,\ V))\ X) \in s \vee$
     $(\mathit{Asset}\ n,\ \mathit{Crypt}\ (\mathit{SesK}\ (x,\ U,\ V))\ (\mathit{Num}\ 0)) \in s)$
    using *C* **by** *auto*
next
  fix *X U K*
  assume $\forall X.\ (\mathit{Spy},\ \langle n,\ X\rangle) \in s \longrightarrow X = \mathit{Pwd}\ n$ **and** $(\mathit{Spy},\ \langle n,\ K\rangle) \in s$
  hence $K = \mathit{Pwd}\ n$ **by** *simp*
  moreover assume $U \in \mathit{key\text{-}sets}\ X\ (\mathit{crypts}\ (\mathit{Log}\ -`\ \mathit{spied}\ s))$
  hence $U \subseteq \mathit{range}\ \mathit{Key}$
    by (*rule key-sets-range*)
  moreover assume $K \in U$
  ultimately show *False* **by** *blast*
qed

proposition *idinfo-pwd*:
$[\![s_0 \models s; \exists X. \langle n, X\rangle \in spied\ s \wedge X \neq Pwd\ n;$
$n \notin bad\text{-}id\text{-}pubkey \cup bad\text{-}id\text{-}shakey]\!] \Longrightarrow$
$\exists SK. SesKey\ SK \in spied\ s \wedge$
$((Owner\ n, Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s \vee$
$(Asset\ n, Crypt\ (SesK\ SK)\ (Num\ 0)) \in s)$
by (*drule rtrancl-start*, *assumption*, *simp*, *blast*, (*erule exE*)+, (*erule conjE*)+, *frule idinfo-pwd-start* [*of - n*], *simp*+, *drule r-into-rtrancl*, *drule rtrancl-trans*, *assumption*, (*drule state-subset*)+, *blast*)

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 $\langle n, Auth\text{-}PriKey\ n\rangle \notin spied\ s$
proof
assume *D*: $\langle n, Auth\text{-}PriKey\ n\rangle \in spied\ s$
hence $n \in bad\text{-}id\text{-}password \cup bad\text{-}id\text{-}pubkey \cup bad\text{-}id\text{-}shakey$
by (*rule contrapos-pp*, *rule-tac idinfo-init* [*OF A*])
moreover have $Auth\text{-}PriKey\ n \in spied\ s$
by (*rule idinfo-msg* [*OF A D*])
hence $n \in bad\text{-}prikey$
by (*rule contrapos-pp*, *rule-tac auth-prikey-secret* [*OF A*])
moreover from *this* **have** *E*: $n \notin bad\text{-}id\text{-}pubkey$
using *B* **by** *simp*
moreover have $n \in bad\text{-}shakey$
proof (*cases* $n \in bad\text{-}id\text{-}shakey$, *simp*)
case *False*
with *D* **and** *E* **have** $\exists SK. SesKey\ SK \in spied\ s \wedge$
$((Owner\ n, Crypt\ (SesK\ SK)\ (Pwd\ n)) \in s \vee$
$(Asset\ n, Crypt\ (SesK\ SK)\ (Num\ 0)) \in s)$
(**is** $\exists SK.\ ?P\ SK \wedge (?Q\ SK \vee ?R\ SK)$)
by (*rule-tac idinfo-pwd* [*OF A*], *rule-tac exI* [*of - Auth-PriKey n*], *simp-all*)
then obtain *SK* **where** *?P SK* **and** *?Q SK* ∨ *?R SK* **by** *blast*
moreover {
assume *?Q SK*
hence $n \in bad\text{-}shakey \cap bad\text{-}prikey$
by (*rule-tac contrapos-pp* [*OF ‹?P SK›*], *rule-tac owner-seskey-secret* [*OF A*])
}
moreover {
assume *?R SK*
hence $n \in bad\text{-}shakey \cap (bad\text{-}pwd \cup bad\text{-}prikey)$
by (*rule-tac contrapos-pp* [*OF ‹?P SK›*], *rule-tac asset-seskey-secret* [*OF A*])
}
ultimately show *?thesis* **by** *blast*

qed
ultimately show *False*
using *C* **by** *blast*
qed

theorem *auth-shakey-anonymous*:
assumes
A: $s_0 \models s$ **and**
B: $n \notin$ *bad-id-shakey* **and**
C: $n \notin$ *bad-shakey* $\cap$ (*bad-id-password* $\cup$ *bad-id-pubkey*)
shows $\langle n,$ *Key* (*Auth-ShaKey n*)$\rangle \notin$ *spied s*
proof
assume *D*: $\langle n,$ *Key* (*Auth-ShaKey n*)$\rangle \in$ *spied s*
hence $n \in$ *bad-id-password* $\cup$ *bad-id-pubkey* $\cup$ *bad-id-shakey*
by (*rule contrapos-pp*, *rule-tac idinfo-init* [*OF A*])
moreover have *Key* (*Auth-ShaKey n*) $\in$ *spied s*
by (*rule idinfo-msg* [*OF A D*])
hence $n \in$ *bad-shakey*
by (*rule contrapos-pp*, *rule-tac auth-shakey-secret* [*OF A*])
ultimately show *False*
using *B* **and** *C* **by** *blast*
qed

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* $\wedge$ *x* = (*insert* (*Asset n*, *PriKey S*) *s* $\cup$
{*Asset n*, *Spy*} $\times$ {*Crypt* (*Auth-ShaKey n*) (*PriKey S*)} $\cup$
{(*Spy*, *Log* (*Crypt* (*Auth-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* ⇒ *stage* ⇒ *stage* ⇒ *bool* **where**
pred-owner-ii n x y ≡ *case x of* (*s*, *S*, -) ⇒
  ∃ *A*. *PriKey A* ∉ *used s* ∧ *y* = (*insert* (*Owner n*, *PriKey A*) *s* ∪
    {*Owner n*, *Spy*} × {⦃*Num 1*, *PubKey A*⦄} ∪
    {*Spy*} × *Log* ‘ {*Crypt* (*Auth-ShaKey n*) (*PriKey S*), ⦃*Num 1*, *PubKey A*⦄},
    S, *A*, *0*, *0*, *0*)

definition *run-owner-ii* :: *agent-id* ⇒ *state* ⇒ *stage* **where**
run-owner-ii n s ≡ *SOME x*. *pred-owner-ii n* (*run-asset-i n s*) *x*

abbreviation *pred-asset-ii* :: *agent-id* ⇒ *stage* ⇒ *stage* ⇒ *bool* **where**
pred-asset-ii n x y ≡ *case x of* (*s*, *S*, *A*, -) ⇒
  ∃ *B*. *PriKey B* ∉ *used s* ∧ *y* = (*insert* (*Asset n*, *PriKey B*) *s* ∪
    {*Asset n*, *Spy*} × {⦃*Num 2*, *PubKey B*⦄} ∪
    {*Spy*} × *Log* ‘ {⦃*Num 1*, *PubKey A*⦄, ⦃*Num 2*, *PubKey B*⦄},
    S, *A*, *B*, *0*, *0*)

definition *run-asset-ii* :: *agent-id* ⇒ *state* ⇒ *stage* **where**
run-asset-ii n s ≡ *SOME x*. *pred-asset-ii n* (*run-owner-ii n s*) *x*

abbreviation *pred-owner-iii* :: *agent-id* ⇒ *stage* ⇒ *stage* ⇒ *bool* **where**
pred-owner-iii n x y ≡ *case x of* (*s*, *S*, *A*, *B*, -) ⇒
  ∃ *C*. *PriKey C* ∉ *used s* ∧ *y* = (*insert* (*Owner n*, *PriKey C*) *s* ∪
    {*Owner n*, *Spy*} × {⦃*Num 3*, *PubKey C*⦄} ∪
    {*Spy*} × *Log* ‘ {⦃*Num 2*, *PubKey B*⦄, ⦃*Num 3*, *PubKey C*⦄},
    S, *A*, *B*, *C*, *0*)

definition *run-owner-iii* :: *agent-id* ⇒ *state* ⇒ *stage* **where**
run-owner-iii n s ≡ *SOME x*. *pred-owner-iii n* (*run-asset-ii n s*) *x*

abbreviation *pred-asset-iii* :: *agent-id* ⇒ *stage* ⇒ *stage* ⇒ *bool* **where**
pred-asset-iii n x y ≡ *case x of* (*s*, *S*, *A*, *B*, *C*, -) ⇒
  ∃ *D*. *PriKey D* ∉ *used s* ∧ *y* = (*insert* (*Asset n*, *PriKey D*) *s* ∪
    {*Asset n*, *Spy*} × {⦃*Num 4*, *PubKey D*⦄} ∪
    {*Spy*} × *Log* ‘ {⦃*Num 3*, *PubKey C*⦄, ⦃*Num 4*, *PubKey D*⦄},
    S, *A*, *B*, *C*, *D*)

definition *run-asset-iii* :: *agent-id* ⇒ *state* ⇒ *stage* **where**
run-asset-iii n s ≡ *SOME x*. *pred-asset-iii n* (*run-owner-iii n s*) *x*

abbreviation *stage-owner-iv* :: *agent-id* ⇒ *stage* ⇒ *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*)} ∪

$\{Spy\} \times Log$ ‘ $\{\{\!|Num\ 4,\ PubKey\ D|\!\},\ Crypt\ (SesK\ SK)\ (PubKey\ D)\}$,
$S,\ A,\ B,\ C,\ D)$

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*
$(s \cup \{Asset\ n\} \times \{SesKey\ SK,\ PubKey\ B\} \cup$
$\{Asset\ n,\ Spy\} \times \{Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK\} \cup$
$\{Spy\} \times Log$ ‘ $\{Crypt\ (SesK\ SK)\ (PubKey\ D)$,
$Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK\}$,
$S,\ A,\ B,\ C,\ D)$

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*
$(s \cup \{Owner\ n,\ Spy\} \times \{Crypt\ (SesK\ SK)\ (Pwd\ n)\} \cup$
$\{Spy\} \times Log$ ‘ $\{Token\ n\ (Auth\text{-}PriK\ n)\ B\ C\ SK,\ Crypt\ (SesK\ SK)\ (Pwd\ n)\}$,
$S,\ A,\ B,\ C,\ D)$

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*
$(s \cup \{Asset\ n,\ Spy\} \times \{Crypt\ (SesK\ SK)\ (Num\ 0)\} \cup$
$\{Spy\} \times Log$ ‘ $\{Crypt\ (SesK\ SK)\ (Pwd\ n),\ Crypt\ (SesK\ SK)\ (Num\ 0)\}$,
$S,\ A,\ B,\ C,\ D)$

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 $\{A.\ PriKey\ A \notin used\ s_0\}$
by (*rule infinite-super* [*of* − *range Auth-PriK*], *rule subsetI*, *simp add*:
image-def bad-prik-def, *rule someI2* [*of* - {}], *simp*, *blast*, *subst Auth-PriK-def*,
rule someI [*of* - λ*n. 0*], *simp*)

lemma *prikey-unused-2*:

$[\![s \vdash s'; \textit{infinite } \{A.\ \textit{PriKey } A \notin \textit{used } s\}]\!] \Longrightarrow$
$\quad \textit{infinite } \{A.\ \textit{PriKey } A \notin \textit{used } s'\}$
by (*simp add*: *rel-def*, ((*erule disjE*)*?*, (*erule exE*)+, *simp add*: *image-iff*, (((*subst conj-commute* | *subst Int-commute*), *simp add*: *Collect-conj-eq Collect-neg-eq Diff-eq* [*symmetric*])+)*?*, ((*rule Diff-infinite-finite*, *rule msg.induct*, *simp-all*, *rule key.induct*, *simp-all*)+)*?*)+)

proposition *prikey-unused*:
$s_0 \models s \Longrightarrow \exists A.\ \textit{PriKey } A \notin \textit{used } s$
by (*subgoal-tac infinite* $\{A.\ \textit{PriKey } A \notin \textit{used } s\}$, *drule infinite-imp-nonempty*, *simp*, *erule rtrancl-induct*, *rule prikey-unused-1*, *rule prikey-unused-2*)

lemma *pubkey-unused-1*:
$[\![s \vdash s'; \textit{PubKey } A \in \textit{parts } (\textit{used } s) \longrightarrow \textit{PriKey } A \in \textit{used } s;$
$\quad \textit{PubKey } A \in \textit{parts } (\textit{used } s')]\!] \Longrightarrow$
$\textit{PriKey } A \in \textit{used } s'$
by (*simp add*: *rel-def*, ((*erule disjE*)*?*, ((*erule exE*)+)*?*, *simp add*: *parts-insert image-iff split*: *if-split-asm*, ((*erule conjE*)+, *drule RangeI*, (*drule parts-used*, *drule parts-snd*)*?*, *simp* | (*subst* (*asm*) *disj-assoc* [*symmetric*])*?*, *erule disjE*, (*drule parts-dec* | *drule parts-enc* | *drule parts-sep* | *drule parts-con*), *simp*)*?*)+)

proposition *pubkey-unused* [*rule-format*]:
$s_0 \models s \Longrightarrow$
$\quad \textit{PriKey } A \notin \textit{used } s \longrightarrow$
$\textit{PubKey } A \notin \textit{parts } (\textit{used } s)$
by (*erule rtrancl-induct*, *subst parts-init*, *simp add*: *Range-iff image-def*, *rule impI*, *erule contrapos-nn* [*OF* - *pubkey-unused-1*], *blast*+)

proposition *run-asset-i-ex*:
$s_0 \models s \Longrightarrow \textit{pred-asset-i } n\ s\ (\textit{run-asset-i } n\ s)$
by (*drule prikey-unused*, *erule exE*, *subst run-asset-i-def*, *rule someI-ex*, *blast*)

proposition *run-asset-i-rel*:
$s_0 \models s \Longrightarrow s \models \textit{fst } (\textit{run-asset-i } n\ s)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*drule run-asset-i-ex* [*of* - *n*], *rule r-into-rtrancl*, *subgoal-tac* (*s*, *?t*) $\in$ *rel-asset-i*, *simp-all add*: *rel-def*, *erule exE*, *auto*)

proposition *run-asset-i-msg*:
$s_0 \models s \Longrightarrow$
$\quad \textit{case run-asset-i } n\ s\ \textit{of } (s', S, \text{-}) \Rightarrow$
$\quad\quad (\textit{Asset } n, \textit{Crypt } (\textit{Auth-ShaKey } n)\ (\textit{PriKey } S)) \in s'$
by (*drule run-asset-i-ex* [*of* - *n*], *auto*)

proposition *run-asset-i-nonce*:
$s_0 \models s \Longrightarrow \textit{PriKey } (\textit{fst } (\textit{snd } (\textit{run-asset-i } n\ s))) \notin \textit{used } s$
by (*drule run-asset-i-ex* [*of* - *n*], *auto*)

proposition *run-asset-i-unused*:
$s_0 \models s \Longrightarrow \exists A.\ PriKey\ A \notin used\ (fst\ (run\text{-}asset\text{-}i\ n\ s))$
by (*rule prikey-unused*, *rule rtrancl-trans*, *simp*, *rule run-asset-i-rel*)

proposition *run-owner-ii-ex*:
$s_0 \models s \Longrightarrow pred\text{-}owner\text{-}ii\ n\ (run\text{-}asset\text{-}i\ n\ s)\ (run\text{-}owner\text{-}ii\ n\ s)$
by (*drule run-asset-i-unused*, *erule exE*, *subst run-owner-ii-def*, *rule someI-ex*, *auto simp add*: *split-def*)

proposition *run-owner-ii-rel*:
$s_0 \models s \Longrightarrow s \models fst\ (run\text{-}owner\text{-}ii\ n\ s)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule run-asset-i-rel* [*of* - *n*], *frule run-asset-i-msg*, *drule run-owner-ii-ex*, *subgoal-tac* (*fst* (*run-asset-i n s*), *?t*) $\in$ *rel-owner-ii*, *simp-all add*: *rel-def split-def*, *erule exE*, (*rule exI*)+, *auto*)

proposition *run-owner-ii-msg*:
$s_0 \models s \Longrightarrow$
case run-owner-ii n s of $(s',\ S,\ A,\ \text{-}) \Rightarrow$
$\{(Asset\ n,\ Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)),$
$(Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\})\} \subseteq s'$
by (*frule run-asset-i-msg* [*of* - *n*], *drule run-owner-ii-ex* [*of* - *n*], *auto*)

proposition *run-owner-ii-nonce*:
$s_0 \models s \Longrightarrow PriKey\ (fst\ (snd\ (run\text{-}owner\text{-}ii\ n\ s))) \notin used\ s$
by (*frule run-asset-i-nonce* [*of* - *n*], *drule run-owner-ii-ex* [*of* - *n*], *auto*)

proposition *run-owner-ii-unused*:
$s_0 \models s \Longrightarrow \exists B.\ PriKey\ B \notin used\ (fst\ (run\text{-}owner\text{-}ii\ n\ s))$
by (*rule prikey-unused*, *rule rtrancl-trans*, *simp*, *rule run-owner-ii-rel*)

proposition *run-asset-ii-ex*:
$s_0 \models s \Longrightarrow pred\text{-}asset\text{-}ii\ n\ (run\text{-}owner\text{-}ii\ n\ s)\ (run\text{-}asset\text{-}ii\ n\ s)$
by (*drule run-owner-ii-unused*, *erule exE*, *subst run-asset-ii-def*, *rule someI-ex*, *auto simp add*: *split-def*)

proposition *run-asset-ii-rel*:
$s_0 \models s \Longrightarrow s \models fst\ (run\text{-}asset\text{-}ii\ n\ s)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule run-owner-ii-rel* [*of* - *n*], *frule run-owner-ii-msg*, *drule run-asset-ii-ex*, *subgoal-tac* (*fst* (*run-owner-ii n s*), *?t*) $\in$ *rel-asset-ii*, *simp-all add*: *rel-def split-def*, *erule exE*, (*rule exI*)+, *auto*)

proposition *run-asset-ii-msg*:
assumes *A*: $s_0 \models s$
shows *case run-asset-ii n s of* $(s',\ S,\ A,\ B,\ \text{-}) \Rightarrow$

$\quad insert\ (Owner\ n,\ \{\!|Num\ 1,\ PubKey\ A|\!\})$
$\quad\ (\{Asset\ n\} \times \{Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S),$
$\quad\ \ \{\!|Num\ 2,\ PubKey\ B|\!\}\}) \subseteq s' \wedge$
$\quad (Asset\ n,\ PubKey\ B) \notin s'$
by (*insert run-owner-ii-msg* [*OF A*, *of n*], *insert run-asset-ii-ex* [*OF A*, *of n*], *simp add*: *split-def*, *erule exE*, *simp*, *insert run-owner-ii-rel* [*OF A*, *of n*], *drule rtrancl-trans* [*OF A*], *drule pubkey-unused*, *auto*)

proposition *run-asset-ii-nonce*:
$s_0 \models s \Longrightarrow PriKey\ (fst\ (snd\ (run\text{-}asset\text{-}ii\ n\ s))) \notin used\ s$
by (*frule run-owner-ii-nonce* [*of - n*], *drule run-asset-ii-ex* [*of - n*], *auto*)

proposition *run-asset-ii-unused*:
$s_0 \models s \Longrightarrow \exists\, C.\ PriKey\ C \notin used\ (fst\ (run\text{-}asset\text{-}ii\ n\ s))$
by (*rule prikey-unused*, *rule rtrancl-trans*, *simp*, *rule run-asset-ii-rel*)

proposition *run-owner-iii-ex*:
$s_0 \models s \Longrightarrow pred\text{-}owner\text{-}iii\ n\ (run\text{-}asset\text{-}ii\ n\ s)\ (run\text{-}owner\text{-}iii\ n\ s)$
by (*drule run-asset-ii-unused*, *erule exE*, *subst run-owner-iii-def*, *rule someI-ex*, *auto simp add*: *split-def*)

proposition *run-owner-iii-rel*:
$s_0 \models s \Longrightarrow s \models fst\ (run\text{-}owner\text{-}iii\ n\ s)$
$\quad$(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule run-asset-ii-rel* [*of - n*], *frule run-asset-ii-msg*, *drule run-owner-iii-ex*, *subgoal-tac* ($fst\ (run\text{-}asset\text{-}ii\ n\ s),\ ?t) \in rel\text{-}owner\text{-}iii$, *simp-all add*: *rel-def split-def*, *erule exE*, (*rule exI*)+, *auto*)

proposition *run-owner-iii-msg*:
$s_0 \models s \Longrightarrow$
$\quad case\ run\text{-}owner\text{-}iii\ n\ s\ of\ (s',\ S,\ A,\ B,\ C,\ \text{-}) \Rightarrow$
$\quad\ \{Asset\ n\} \times \{Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S),\ \{\!|Num\ 2,\ PubKey\ B|\!\}\} \cup$
$\quad\ \{Owner\ n\} \times \{\{\!|Num\ 1,\ PubKey\ A|\!\},\ \{\!|Num\ 3,\ PubKey\ C|\!\}\} \subseteq s' \wedge$
$\quad\ (Asset\ n,\ PubKey\ B) \notin s'$
by (*frule run-asset-ii-msg* [*of - n*], *drule run-owner-iii-ex* [*of - n*], *auto*)

proposition *run-owner-iii-nonce*:
$s_0 \models s \Longrightarrow PriKey\ (fst\ (snd\ (run\text{-}owner\text{-}iii\ n\ s))) \notin used\ s$
by (*frule run-asset-ii-nonce* [*of - n*], *drule run-owner-iii-ex* [*of - n*], *auto*)

proposition *run-owner-iii-unused*:
$s_0 \models s \Longrightarrow \exists\, D.\ PriKey\ D \notin used\ (fst\ (run\text{-}owner\text{-}iii\ n\ s))$
by (*rule prikey-unused*, *rule rtrancl-trans*, *simp*, *rule run-owner-iii-rel*)

proposition *run-asset-iii-ex*:
$s_0 \models s \Longrightarrow pred\text{-}asset\text{-}iii\ n\ (run\text{-}owner\text{-}iii\ n\ s)\ (run\text{-}asset\text{-}iii\ n\ s)$
by (*drule run-owner-iii-unused*, *erule exE*, *subst run-asset-iii-def*, *rule someI-ex*,

auto simp add: *split-def*)

proposition *run-asset-iii-rel*:
$s_0 \models s \Longrightarrow s \models fst\ (run\text{-}asset\text{-}iii\ n\ s)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule run-owner-iii-rel* [*of* - *n*], *frule run-owner-iii-msg*, *drule run-asset-iii-ex*, *subgoal-tac* (*fst* (*run-owner-iii n s*), *?t*) $\in$ *rel-asset-iii*, *simp-all add*: *rel-def split-def*, *erule exE*, (*rule exI*)+, *auto*)

proposition *run-asset-iii-msg*:
$s_0 \models s \Longrightarrow$
case run-asset-iii n s of $(s', S, A, B, C, D) \Rightarrow$
$\{Asset\ n\} \times \{Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S),\ \{|Num\ 2,\ PubKey\ B|\},$
$\{|Num\ 4,\ PubKey\ D|\}\} \cup$
$\{Owner\ n\} \times \{\{|Num\ 1,\ PubKey\ A|\},\ \{|Num\ 3,\ PubKey\ C|\}\} \subseteq s' \wedge$
$(Asset\ n,\ PubKey\ B) \notin s'$
by (*frule run-owner-iii-msg* [*of* - *n*], *drule run-asset-iii-ex* [*of* - *n*], *auto*)

proposition *run-asset-iii-nonce*:
$s_0 \models s \Longrightarrow PriKey\ (fst\ (snd\ (run\text{-}asset\text{-}iii\ n\ s))) \notin used\ s$
by (*frule run-owner-iii-nonce* [*of* - *n*], *drule run-asset-iii-ex* [*of* - *n*], *auto*)

lemma *run-owner-iv-rel-1*:
$[\![s_0 \models s;\ run\text{-}asset\text{-}iii\ n\ s = (s', S, A, B, C, D)]\!] \Longrightarrow$
$s \models fst\ (run\text{-}owner\text{-}iv\ n\ s)$
(**is** $[\![$-; -$]\!]$ $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule run-asset-iii-rel* [*of* - *n*], *drule run-asset-iii-msg* [*of* - *n*], *subgoal-tac* (*s'*, *?t*) $\in$ *rel-owner-iv*, *simp-all add*: *rel-def run-owner-iv-def Let-def*, *rule exI* [*of* - *n*], *rule exI* [*of* - *S*], *rule exI* [*of* - *A*], *rule exI* [*of* - *B*], *rule exI* [*of* - *C*], *rule exI* [*of* - *D*], *rule exI* [*of* - *Auth-ShaKey n*], *auto*)

proposition *run-owner-iv-rel*:
$s_0 \models s \Longrightarrow s \models fst\ (run\text{-}owner\text{-}iv\ n\ s)$
by (*insert run-owner-iv-rel-1*, *cases run-asset-iii n s*, *simp*)

proposition *run-owner-iv-msg*:
$s_0 \models s \Longrightarrow$
let $(s', S, A, B, C, D) = run\text{-}owner\text{-}iv\ n\ s;$
$SK = (Some\ S,\ \{A, B\},\ \{C, D\})$ *in*
$\{Asset\ n\} \times \{Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S),\ \{|Num\ 2,\ PubKey\ B|\},$
$\{|Num\ 4,\ PubKey\ D|\}\} \cup$
$\{Owner\ n\} \times \{\{|Num\ 1,\ PubKey\ A|\},\ \{|Num\ 3,\ PubKey\ C|\},\ SesKey\ SK,$
$Crypt\ (SesK\ SK)\ (PubKey\ D)\} \subseteq s' \wedge$
$(Asset\ n,\ PubKey\ B) \notin s'$
by (*drule run-asset-iii-msg* [*of* - *n*], *simp add*: *run-owner-iv-def split-def Let-def*)

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

by (*drule run-asset-iii-nonce* [*of* - *n*], *simp add*: *run-owner-iv-def split-def Let-def*)

proposition *run-asset-iv-rel*:
$s_0 \models s \Longrightarrow s \models$ *fst* (*run-asset-iv n s*)
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule run-owner-iv-rel* [*of* - *n*], *drule run-owner-iv-msg* [*of* - *n*], *subgoal-tac* (*fst* (*run-owner-iv n s*), *?t*) $\in$ *rel-asset-iv*, *simp-all add*: *rel-def run-asset-iv-def split-def Let-def*, *blast*)

proposition *run-asset-iv-msg*:
$s_0 \models s \Longrightarrow$
let (s', *S*, *A*, *B*, *C*, *D*) = *run-asset-iv n s*; *SK* = (*Some S*, {*A*, *B*}, {*C*, *D*}) *in*
insert (*Owner n*, *SesKey SK*)
({*Asset n*} $\times$ {*SesKey SK*, *Token n* (*Auth-PriK n*) *B C SK*}) $\subseteq s'$
by (*drule run-owner-iv-msg* [*of* - *n*], *simp add*: *run-asset-iv-def split-def Let-def*)

proposition *run-asset-iv-nonce*:
$s_0 \models s \Longrightarrow$ *PriKey* (*fst* (*snd* (*run-asset-iv n s*))) $\notin$ *used s*
by (*drule run-owner-iv-nonce* [*of* - *n*], *simp add*: *run-asset-iv-def split-def Let-def*)

proposition *run-owner-v-rel*:
$s_0 \models s \Longrightarrow s \models$ *fst* (*run-owner-v n s*)
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule run-asset-iv-rel* [*of* - *n*], *drule run-asset-iv-msg* [*of* - *n*], *subgoal-tac* (*fst* (*run-asset-iv n s*), *?t*) $\in$ *rel-owner-v*, *simp-all add*: *rel-def run-owner-v-def split-def Let-def*, *blast*)

proposition *run-owner-v-msg*:
$s_0 \models s \Longrightarrow$
let (s', *S*, *A*, *B*, *C*, *D*) = *run-owner-v n s*;
SK = (*Some S*, {*A*, *B*}, {*C*, *D*}) *in*
{(*Asset n*, *SesKey SK*),
(*Owner n*, *Crypt* (*SesK SK*) (*Pwd n*))} $\subseteq s'$
by (*drule run-asset-iv-msg* [*of* - *n*], *simp add*: *run-owner-v-def split-def Let-def*)

proposition *run-owner-v-nonce*:
$s_0 \models s \Longrightarrow$ *PriKey* (*fst* (*snd* (*run-owner-v n s*))) $\notin$ *used s*
by (*drule run-asset-iv-nonce* [*of* - *n*], *simp add*: *run-owner-v-def split-def Let-def*)

proposition *run-asset-v-rel*:
$s_0 \models s \Longrightarrow s \models$ *fst* (*run-asset-v n s*)
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule run-owner-v-rel* [*of* - *n*], *drule run-owner-v-msg* [*of* - *n*], *subgoal-tac* (*fst* (*run-owner-v n s*), *?t*) $\in$ *rel-asset-v*, *simp-all add*: *rel-def run-asset-v-def split-def Let-def*, *blast*)

proposition *run-asset-v-msg*:
$s_0 \models s \Longrightarrow$
  let $(s', S, A, B, C, D) =$ *run-asset-v n s*; $SK = (Some\ S, \{A, B\}, \{C, D\})$ *in*
  $\{(Owner\ n, Crypt\ (SesK\ SK)\ (Pwd\ n)),$
  $(Asset\ n, Crypt\ (SesK\ SK)\ (Num\ 0))\} \subseteq s'$
by (*drule run-owner-v-msg* [*of - n*], *simp add*: *run-asset-v-def split-def Let-def*)

proposition *run-asset-v-nonce*:
$s_0 \models s \Longrightarrow PriKey\ (fst\ (snd\ (run\text{-}asset\text{-}v\ n\ s))) \notin used\ s$
by (*drule run-owner-v-nonce* [*of - n*], *simp add*: *run-asset-v-def split-def Let-def*)

lemma *runs-unbounded-1*:
$[\![s_0 \models s;\ run\text{-}asset\text{-}v\ n\ s = (s', S, A, B, C, D)]\!] \Longrightarrow$
  $\exists s'\ S\ SK.\ (Asset\ n, Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \notin s \wedge$
  $\{(Owner\ n, Crypt\ (SesK\ SK)\ (Pwd\ n)),$
  $(Asset\ n, Crypt\ (SesK\ SK)\ (Num\ 0))\} \subseteq s' \wedge$
  $s \models s' \wedge fst\ SK = Some\ S$
by (*rule exI* [*of - s'*], *rule exI* [*of - S*], *rule exI* [*of -* (*Some S*, $\{A, B\}$, $\{C, D\}$)],
rule conjI, *rule notI*, *frule run-asset-v-nonce* [*of - n*], *frule asset-i-used* [*of - n S*],
simp, *frule run-asset-v-rel* [*of - n*], *drule run-asset-v-msg* [*of - n*],
simp add: *Let-def*)

theorem *runs-unbounded*:
$s_0 \models s \Longrightarrow \exists s'\ S\ SK.\ s \models s' \wedge fst\ SK = Some\ S \wedge$
  $(Asset\ n, Crypt\ (Auth\text{-}ShaKey\ n)\ (PriKey\ S)) \notin s \wedge$
  $\{(Owner\ n, Crypt\ (SesK\ SK)\ (Pwd\ n)),$
  $(Asset\ n, Crypt\ (SesK\ SK)\ (Num\ 0))\} \subseteq s'$
by (*insert runs-unbounded-1*, *cases run-asset-v n s*, *blast*)

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

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

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

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

definition *pwd-spy-sesk* :: *agent-id* ⇒ *stage* **where**
pwd-spy-sesk n ≡
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* ⇒ *stage* **where**
pwd-spy-mult n ≡
case pwd-spy-sesk n of (*s*, *S*, *A*, *B*, *C*, *D*) ⇒
(*insert* (*Spy*, *Auth-PriK n* ⊗ *B*) *s*, *S*, *A*, *B*, *C*, *D*)

definition *pwd-spy-enc-pubk* :: *agent-id* ⇒ *stage* **where**
pwd-spy-enc-pubk n ≡
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* $\otimes$ *B*),
Crypt (*SesK SK*) (*Sign n* (*Auth-PriK n*))$|\!\}$) *s*, *S*, *A*, *B*, *C*, *D*)

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

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

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

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

definition *pwd-spy-id-sesk* :: *agent-id* $\Rightarrow$ *stage* **where**
pwd-spy-id-sesk n $\equiv$
let (*s*, *S*, *A*, *B*, *C*, *D*) = *pwd-spy-id-pubk n*;
SK = (*Some S*, {*A*, *B*}, {*C*, *D*}) *in*
(*insert* (*Spy*, $\langle$*n*, *SesKey SK*$\rangle$) *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*:
$[\![$*U* $\in$ *key-sets X* (*crypts* (*Log* −‘ *spied H*)); *H* $\subseteq$ *H'*$]\!]$ $\Longrightarrow$
U $\in$ *key-sets X* (*crypts* (*Log* −‘ *spied H'*))
(**is** $[\![$- $\in$ *?A*; -$]\!]$ $\Longrightarrow$ -)
by (*rule subsetD* [*of ?A*], *rule key-sets-mono*, *rule crypts-mono*, *blast*)

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*),

$\langle n,\ Key\ (Auth\text{-}ShaKey\ n)\rangle\} \cup range\ Num)$

proposition *pwd-spy-i-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}i\ n)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule r-into-rtrancl*, *subgoal-tac* (s_0, *?t*) $\in$ *rel-enc*, *simp-all add*: *rel-def pwd-spy-i-def*, *blast*)

proposition *pwd-spy-i-msg*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
case pwd-spy-i n of $(s,\ S,\ A,\ B,\ C,\ D) \Rightarrow$
pwd-spy-i-state n $(S,\ A,\ B,\ C,\ D) \subseteq s$
by (*simp add*: *pwd-spy-i-def*, *blast*)

proposition *pwd-spy-i-unused*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow \exists A.\ PriKey\ A \notin used\ (fst\ (pwd\text{-}spy\text{-}i\ n))$
by (*drule pwd-spy-i-rel*, *rule prikey-unused*)

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*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
pred-owner-ii n (*pwd-spy-i n*) (*pwd-owner-ii n*)
by (*drule pwd-spy-i-unused*, *erule exE*, *subst pwd-owner-ii-def*, *rule someI-ex*, *auto simp add*: *split-def*)

proposition *pwd-owner-ii-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}owner\text{-}ii\ n)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-i-rel*, *frule pwd-spy-i-msg*, *drule pwd-owner-ii-ex*, *subgoal-tac* (*fst* (*pwd-spy-i n*), *?t*) $\in$ *rel-owner-ii*, *simp-all add*: *rel-def split-def*, *erule exE*, *rule exI*, *auto*)

proposition *pwd-owner-ii-msg*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
case pwd-owner-ii n of $(s,\ S,\ A,\ B,\ C,\ D) \Rightarrow$
pwd-owner-ii-state n $(S,\ A,\ B,\ C,\ D) \subseteq s\ \wedge$
$\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -`\ spied\ s))$
by (*frule pwd-spy-i-msg*, *drule pwd-owner-ii-ex*, *simp add*: *split-def*, *erule exE*, *simp add*: *Image-def*, *simp only*: *Collect-disj-eq crypts-union key-sets-union*, *simp add*: *crypts-insert key-sets-insert*, *blast*)

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-spy-ii n*)
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-owner-ii-rel*, *drule pwd-owner-ii-msg*, *subgoal-tac* (*fst* (*pwd-owner-ii n*), *?t*) $\in$ *rel-con*, *simp-all add*: *rel-def pwd-spy-ii-def split-def*, *blast*)

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-ShaKey n*)$\}$ $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* −‘ *spied s*))
by (*drule pwd-owner-ii-msg*, *simp add*: *pwd-spy-ii-def split-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

proposition *pwd-spy-ii-unused*:
$n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow \exists\, C$. *PriKey C* $\notin$ *used* (*fst* (*pwd-spy-ii n*))
by (*drule pwd-spy-ii-rel*, *rule prikey-unused*)

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-spy-ii n*) (*pwd-owner-iii n*)
by (*drule pwd-spy-ii-unused*, *erule exE*, *subst pwd-owner-iii-def*, *rule someI-ex*, *auto simp add*: *split-def*)

proposition *pwd-owner-iii-rel*:
$n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* (*pwd-owner-iii n*)
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-ii-rel*, *frule pwd-spy-ii-msg*, *drule pwd-owner-iii-ex*, *subgoal-tac* (*fst* (*pwd-spy-ii n*), *?t*) $\in$ *rel-owner-iii*, *simp-all add*: *rel-def split-def*, *rule exI*, *rule exI*, *auto*)

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-ShaKey n*)$\}$ $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* −‘ *spied s*))
by (*frule pwd-spy-ii-msg*, *drule pwd-owner-iii-ex*, *simp add*: *split-def*, *erule exE*, *simp*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-iii-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *state* **where**

pwd-spy-iii-state n $(S, A, B, C, D) =$
pwd-owner-iii-state n $(S, A, B, C, D) \cup \{Spy\} \times \{PriKey\ D,$
$\{\!|Num\ 4,\ PubKey\ D|\!\}\}$

proposition *pwd-spy-iii-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}iii\ n)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-owner-iii-rel*, *drule pwd-owner-iii-msg*, *subgoal-tac* (*fst* (*pwd-owner-iii n*), *?t*) $\in$ *rel-con*, *simp-all add*: *rel-def pwd-spy-iii-def split-def*, *blast*)

proposition *pwd-spy-iii-msg*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
case pwd-spy-iii n of $(s, S, A, B, C, D) \Rightarrow$
pwd-spy-iii-state n $(S, A, B, C, D) \subseteq s \wedge$
$\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-owner-iii-msg*, *simp add*: *pwd-spy-iii-def split-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-owner-iv-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *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*:
$[\![n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey;\ pwd\text{-}spy\text{-}iii\ n = (s, S, A, B, C, D)]\!] \Longrightarrow$
$s_0 \models fst\ (pwd\text{-}owner\text{-}iv\ n)$
(**is** $[\![$-; -$]\!] \Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-iii-rel*, *drule pwd-spy-iii-msg*, *subgoal-tac* (*s*, *?t*) $\in$ *rel-owner-iv*, *simp-all add*: *rel-def pwd-owner-iv-def Let-def*, *rule exI* [*of* - *n*], *rule exI* [*of* - *S*], *rule exI* [*of* - *A*], *rule exI* [*of* - *B*], *rule exI* [*of* - *C*], *rule exI* [*of* - *D*], *rule exI* [*of* - *Auth-ShaKey n*], *auto*)

proposition *pwd-owner-iv-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}owner\text{-}iv\ n)$
by (*insert pwd-owner-iv-rel-1*, *cases pwd-spy-iii n*, *simp*)

proposition *pwd-owner-iv-msg*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
case pwd-owner-iv n of $(s, S, A, B, C, D) \Rightarrow$
pwd-owner-iv-state n $(S, A, B, C, D) \subseteq s \wedge$
$\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-spy-iii-msg*, *simp add*: *pwd-owner-iv-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-sep-map-state* :: *agent-id* $\Rightarrow$ *seskey-tuple* $\Rightarrow$ *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*:
$n \in$ *bad-prikey* $\cap$ *bad-id-shakey* $\Longrightarrow s_0 \models$ *fst* (*pwd-spy-sep-map n*)
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-owner-iv-rel*, *drule pwd-owner-iv-msg*, *subgoal-tac* (*fst* (*pwd-owner-iv n*), *?t*) $\in$ *rel-sep*, *simp-all add*: *rel-def pwd-spy-sep-map-def split-def*, *blast*)

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) $\subseteq s \land$
$\{$*Key* (*Auth-ShaKey n*)$\}$ $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* $-$‘ *spied s*))
by (*drule pwd-owner-iv-msg*, *simp add*: *pwd-spy-sep-map-def split-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

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-spy-sep-map-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-spy-sep-agr n*)
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-sep-map-rel*, *drule pwd-spy-sep-map-msg*, *subgoal-tac* (*fst* (*pwd-spy-sep-map n*), *?t*) $\in$ *rel-sep*, *simp-all add*: *rel-def pwd-spy-sep-agr-def split-def*, *blast*)

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) $\subseteq s \land$
$\{$*Key* (*Auth-ShaKey n*)$\}$ $\in$ *key-sets* (*PriKey S*) (*crypts* (*Log* $-$‘ *spied s*))
by (*drule pwd-spy-sep-map-msg*, *simp add*: *pwd-spy-sep-agr-def split-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

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-spy-sep-agr-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-spy-sesk n*)
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-sep-agr-rel*, *drule pwd-spy-sep-agr-msg*, *subgoal-tac* (*fst* (*pwd-spy-sep-agr n*), *?t*) $\in$ *rel-sesk*, *simp-all add*: *rel-def pwd-spy-sesk-def split-def Let-def*, *blast*)

proposition *pwd-spy-sesk-msg*:

$n \in \textit{bad-prikey} \cap \textit{bad-id-shakey} \Longrightarrow$
  case pwd-spy-sesk n of $(s, S, A, B, C, D) \Rightarrow$
    pwd-spy-sesk-state n $(S, A, B, C, D) \subseteq s \wedge$
    $\{\textit{Key}\ (\textit{Auth-ShaKey}\ n)\} \in \textit{key-sets}\ (\textit{PriKey}\ S)\ (\textit{crypts}\ (\textit{Log}\ -`\ \textit{spied}\ s))$
by (*drule pwd-spy-sep-agr-msg*, *simp add*: *pwd-spy-sesk-def split-def Let-def*,
(*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-mult-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *state* **where**
pwd-spy-mult-state n $(S, A, B, C, D) =$
  insert (*Spy*, *Auth-PriK n* ⊗ *B*) (*pwd-spy-sesk-state n* (S, A, B, C, D))

proposition *pwd-spy-mult-rel*:
$n \in \textit{bad-prikey} \cap \textit{bad-id-shakey} \Longrightarrow s_0 \models \textit{fst}\ (\textit{pwd-spy-mult}\ n)$
  (**is** - ⟹ - ⊨ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-sesk-rel*, *drule pwd-spy-sesk-msg*,
subgoal-tac (*fst* (*pwd-spy-sesk n*), *?t*) ∈ *rel-mult*, *simp-all add*: *rel-def*
pwd-spy-mult-def split-def, *blast*)

proposition *pwd-spy-mult-msg*:
$n \in \textit{bad-prikey} \cap \textit{bad-id-shakey} \Longrightarrow$
  case pwd-spy-mult n of $(s, S, A, B, C, D) \Rightarrow$
    pwd-spy-mult-state n $(S, A, B, C, D) \subseteq s \wedge$
    $\{\textit{Key}\ (\textit{Auth-ShaKey}\ n)\} \in \textit{key-sets}\ (\textit{PriKey}\ S)\ (\textit{crypts}\ (\textit{Log}\ -`\ \textit{spied}\ s))$
by (*drule pwd-spy-sesk-msg*, *simp add*: *pwd-spy-mult-def split-def*,
(*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-enc-pubk-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *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*:
$n \in \textit{bad-prikey} \cap \textit{bad-id-shakey} \Longrightarrow s_0 \models \textit{fst}\ (\textit{pwd-spy-enc-pubk}\ n)$
  (**is** - ⟹ - ⊨ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-mult-rel*, *drule pwd-spy-mult-msg*,
subgoal-tac (*fst* (*pwd-spy-mult n*), *?t*) ∈ *rel-enc*, *simp-all add*: *rel-def*
pwd-spy-enc-pubk-def split-def Let-def, *blast*)

proposition *pwd-spy-enc-pubk-msg*:
$n \in \textit{bad-prikey} \cap \textit{bad-id-shakey} \Longrightarrow$
  case pwd-spy-enc-pubk n of $(s, S, A, B, C, D) \Rightarrow$
    pwd-spy-enc-pubk-state n $(S, A, B, C, D) \subseteq s \wedge$
    $\{\textit{Key}\ (\textit{Auth-ShaKey}\ n)\} \in \textit{key-sets}\ (\textit{PriKey}\ S)\ (\textit{crypts}\ (\textit{Log}\ -`\ \textit{spied}\ s))$
by (*drule pwd-spy-mult-msg*, *simp add*: *pwd-spy-enc-pubk-def split-def Let-def*,
(*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-enc-mult-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *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* ⊗ *B*))
(*pwd-spy-enc-pubk-state n* (*S*, *A*, *B*, *C*, *D*)))

proposition *pwd-spy-enc-mult-rel*:
$n \in$ *bad-prikey* ∩ *bad-id-shakey* ⟹ s_0 ⊨ *fst* (*pwd-spy-enc-mult n*)
(**is** - ⟹ - ⊨ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-enc-pubk-rel*, *drule pwd-spy-enc-pubk-msg*, *subgoal-tac* (*fst* (*pwd-spy-enc-pubk n*), *?t*) ∈ *rel-enc*, *simp-all add*: *rel-def pwd-spy-enc-mult-def split-def Let-def*, *blast*)

proposition *pwd-spy-enc-mult-msg*:
$n \in$ *bad-prikey* ∩ *bad-id-shakey* ⟹
case pwd-spy-enc-mult n of (*s*, *S*, *A*, *B*, *C*, *D*) ⇒
pwd-spy-enc-mult-state n (*S*, *A*, *B*, *C*, *D*) ⊆ *s* ∧
{*Key* (*Auth-ShaKey n*)} ∈ *key-sets* (*PriKey S*) (*crypts* (*Log* −‘ *spied s*))
by (*drule pwd-spy-enc-pubk-msg*, *simp add*: *pwd-spy-enc-mult-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-enc-sign-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *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-PriK n*)))
(*pwd-spy-enc-mult-state n* (*S*, *A*, *B*, *C*, *D*)))

proposition *pwd-spy-enc-sign-rel*:
$n \in$ *bad-prikey* ∩ *bad-id-shakey* ⟹ s_0 ⊨ *fst* (*pwd-spy-enc-sign n*)
(**is** - ⟹ - ⊨ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-enc-mult-rel*, *drule pwd-spy-enc-mult-msg*, *subgoal-tac* (*fst* (*pwd-spy-enc-mult n*), *?t*) ∈ *rel-enc*, *simp-all add*: *rel-def pwd-spy-enc-sign-def split-def Let-def*, *blast*)

proposition *pwd-spy-enc-sign-msg*:
$n \in$ *bad-prikey* ∩ *bad-id-shakey* ⟹
case pwd-spy-enc-sign n of (*s*, *S*, *A*, *B*, *C*, *D*) ⇒
pwd-spy-enc-sign-state n (*S*, *A*, *B*, *C*, *D*) ⊆ *s* ∧
{*Key* (*Auth-ShaKey n*)} ∈ *key-sets* (*PriKey S*) (*crypts* (*Log* −‘ *spied s*))
by (*drule pwd-spy-enc-mult-msg*, *simp add*: *pwd-spy-enc-sign-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-con-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *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-PriK n* ⊗ *B*),
Crypt (*SesK SK*) (*Sign n* (*Auth-PriK n*))⦄)

$(pwd\text{-}spy\text{-}enc\text{-}sign\text{-}state\ n\ (S,\ A,\ B,\ C,\ D)))$

proposition *pwd-spy-con-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}con\ n)$
$(\mathbf{is}\ \text{-} \Longrightarrow \text{-} \models\ ?t)$
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-enc-sign-rel*, *drule pwd-spy-enc-sign-msg*, *subgoal-tac* (*fst* (*pwd-spy-enc-sign n*), *?t*) ∈ *rel-con*, *simp-all add*: *rel-def pwd-spy-con-def split-def Let-def*, *blast*)

proposition *pwd-spy-con-msg*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
$case\ pwd\text{-}spy\text{-}con\ n\ of\ (s,\ S,\ A,\ B,\ C,\ D) \Rightarrow$
$pwd\text{-}spy\text{-}con\text{-}state\ n\ (S,\ A,\ B,\ C,\ D) \subseteq s\ \wedge$
$\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-spy-enc-sign-msg*, *simp add*: *pwd-spy-con-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

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

proposition *pwd-spy-iv-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}iv\ n)$
$(\mathbf{is}\ \text{-} \Longrightarrow \text{-} \models\ ?t)$
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-con-rel*, *drule pwd-spy-con-msg*, *subgoal-tac* (*fst* (*pwd-spy-con n*), *?t*) ∈ *rel-con*, *simp-all add*: *rel-def pwd-spy-iv-def split-def Let-def*, *blast*)

proposition *pwd-spy-iv-msg*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
$case\ pwd\text{-}spy\text{-}iv\ n\ of\ (s,\ S,\ A,\ B,\ C,\ D) \Rightarrow$
$pwd\text{-}spy\text{-}iv\text{-}state\ n\ (S,\ A,\ B,\ C,\ D) \subseteq s\ \wedge$
$\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-spy-con-msg*, *simp add*: *pwd-spy-iv-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

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

proposition *pwd-owner-v-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}owner\text{-}v\ n)$
$(\mathbf{is}\ \text{-} \Longrightarrow \text{-} \models\ ?t)$
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-iv-rel*, *drule pwd-spy-iv-msg*, *subgoal-tac* (*fst* (*pwd-spy-iv n*), *?t*) ∈ *rel-owner-v*, *simp-all add*: *rel-def pwd-owner-v-def split-def Let-def*, (*rule exI*)+, *blast*)

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 $(S,\ A,\ B,\ C,\ D) \subseteq s \wedge$
$\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -`\ spied\ s)) \wedge$
$\{SesKey\ SK\} \in key\text{-}sets\ (Pwd\ n)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-spy-iv-msg*, *simp add*: *pwd-owner-v-def split-def Let-def*, (*erule conjE*)+, (*rule conjI*, (*erule key-sets-crypts-subset*)?, *blast*)+, *simp add*: *Image-def*, *simp only*: *Collect-disj-eq crypts-union key-sets-union*, *simp add*: *crypts-insert key-sets-insert*)

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*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-owner-v-rel*, *drule pwd-owner-v-msg*, *subgoal-tac* (*fst* (*pwd-owner-v n*), *?t*) $\in$ *rel-dec*, *simp-all add*: *rel-def pwd-spy-dec-def split-def Let-def*, (*rule exI*)+, *auto*)

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 $(S,\ A,\ B,\ C,\ D) \subseteq s \wedge$
$\{Key\ (Auth\text{-}ShaKey\ n)\} \in key\text{-}sets\ (PriKey\ S)\ (crypts\ (Log\ -`\ spied\ s)) \wedge$
$\{SesKey\ SK\} \in key\text{-}sets\ (Pwd\ n)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-owner-v-msg*, *simp add*: *pwd-spy-dec-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI*)?, (*erule key-sets-crypts-subset*)?, *blast*)+)

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*, $\langle n,\ PriKey\ S\rangle$) (*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*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-dec-rel*, *drule pwd-spy-dec-msg*, *subgoal-tac* (*fst* (*pwd-spy-dec n*), *?t*) $\in$ *rel-id-crypt*, *simp-all add*: *rel-def pwd-spy-id-prik-def split-def Let-def*, (*rule exI*)+, *blast*)

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 $(S,\ A,\ B,\ C,\ D) \subseteq s \wedge$

$\{SesKey\ SK\} \in key\text{-}sets\ (Pwd\ n)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-spy-dec-msg*, *simp add*: *pwd-spy-id-prik-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-id-pubk-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *state* **where**
pwd-spy-id-pubk-state n (*S*, *A*, *B*, *C*, *D*) =
insert (*Spy*, ⟨*n*, *PubKey S*⟩) (*pwd-spy-id-prik-state n* (*S*, *A*, *B*, *C*, *D*))

proposition *pwd-spy-id-pubk-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}id\text{-}pubk\ n)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-id-prik-rel*, *drule pwd-spy-id-prik-msg*, *subgoal-tac* (*fst* (*pwd-spy-id-prik n*), *?t*) ∈ *rel-id-invk*, *simp-all add*: *rel-def pwd-spy-id-pubk-def split-def Let-def*, (*rule exI*)+, *auto*)

proposition *pwd-spy-id-pubk-msg*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
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*) ⊆ *s* ∧
$\{SesKey\ SK\} \in key\text{-}sets\ (Pwd\ n)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-spy-id-prik-msg*, *simp add*: *pwd-spy-id-pubk-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

fun *pwd-spy-id-sesk-state* :: *agent-id* ⇒ *seskey-tuple* ⇒ *state* **where**
pwd-spy-id-sesk-state n (*S*, *A*, *B*, *C*, *D*) =
(*let SK* = (*Some S*, {*A*, *B*}, {*C*, *D*}) *in*
insert (*Spy*, ⟨*n*, *SesKey SK*⟩) (*pwd-spy-id-pubk-state n* (*S*, *A*, *B*, *C*, *D*)))

proposition *pwd-spy-id-sesk-rel*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow s_0 \models fst\ (pwd\text{-}spy\text{-}id\text{-}sesk\ n)$
(**is** - $\Longrightarrow$ - $\models$ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-id-pubk-rel*, *drule pwd-spy-id-pubk-msg*, *subgoal-tac* (*fst* (*pwd-spy-id-pubk n*), *?t*) ∈ *rel-id-sesk*, *simp-all add*: *rel-def pwd-spy-id-sesk-def split-def Let-def*, *rule exI*, *rule exI*, *rule exI* [*of - Some* (*fst* (*snd* (*pwd-spy-id-pubk n*)))], *auto*)

proposition *pwd-spy-id-sesk-msg*:
$n \in bad\text{-}prikey \cap bad\text{-}id\text{-}shakey \Longrightarrow$
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*) ⊆ *s* ∧
$\{SesKey\ SK\} \in key\text{-}sets\ (Pwd\ n)\ (crypts\ (Log\ -`\ spied\ s))$
by (*drule pwd-spy-id-pubk-msg*, *simp add*: *pwd-spy-id-sesk-def split-def Let-def*, (*erule conjE*)+, ((*rule conjI* | *erule key-sets-crypts-subset*), *blast*)+)

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*:
$n \in$ *bad-prikey* ∩ *bad-id-shakey* ⟹ s_0 ⊨ *fst* (*pwd-spy-id-pwd n*)
(**is** - ⟹ - ⊨ *?t*)
by (*rule rtrancl-into-rtrancl*, *erule pwd-spy-id-sesk-rel*, *drule pwd-spy-id-sesk-msg*, *subgoal-tac* (*fst* (*pwd-spy-id-sesk n*), *?t*) ∈ *rel-id-crypt*, *simp-all add*: *rel-def pwd-spy-id-pwd-def split-def Let-def*, (*rule exI*)+, *blast*)

proposition *pwd-spy-id-pwd-msg*:
$n \in$ *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*
by (*drule pwd-spy-id-sesk-msg*, *simp add*: *pwd-spy-id-pwd-def split-def Let-def*, *blast*)

theorem *pwd-compromised*:
$n \in$ *bad-prikey* ∩ *bad-id-shakey* ⟹ ∃ *s*. s_0 ⊨ *s* ∧ {*Pwd n*, ⟨*n*, *Pwd n*⟩} ⊆ *spied s*
by (*rule exI* [*of* - *fst* (*pwd-spy-id-pwd n*)], *rule conjI*, *erule pwd-spy-id-pwd-rel*, *drule pwd-spy-id-pwd-msg*, *simp add*: *split-def*)

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 protocols. *Journal of Computer Security*, Dec. 1998.