Verification of a Diffie-Hellman Password-based Authentication Protocol by Extending the Inductive Method

Pasquale Noce
Security Certification Specialist at Arjo Systems, Italy
pasquale dot noce dot lavoro at gmail dot com
pasquale dot noce at arjosystems dot com

September 13, 2023

Abstract

This paper constructs a formal model of a Diffie-Hellman password-based authentication protocol between a user and a smart card, and proves its security. The protocol provides for the dispatch of the user's password to the smart card on a secure messaging channel established by means of Password Authenticated Connection Establishment (PACE), where the mapping method being used is Chip Authentication Mapping. By applying and suitably extending Paulson's Inductive Method, this paper proves that the protocol establishes trustworthy secure messaging channels, preserves the secrecy of users' passwords, and provides an effective mutual authentication service. What is more, these security properties turn out to hold independently of the secrecy of the PACE authentication key.

Contents

1 Propaedeutic definitions and lemmas

theory *Propaedeutics*
imports *Complex-Main HOL−Library.Countable*
begin

declare [[*goals-limit* = *20*]]

This paper is an achievement of the whole OS Development and Certification team of the Arjo Systems site at Arzano, Italy, because it would have never been born without the contributions of my colleagues, the discussions we had, the ideas they shared with me. Particularly, the intuition that the use of Chip Authentication Mapping makes the secrecy of the PACE authentication key unnecessary is not mine. I am very grateful to all the team members for these essential contributions, and even more for these unforgettable years of work together.

1.1 Introduction

Password-based authentication in an insecure environment – such as password-based authentication between a user and a smart card, which is the subject of this paper – requires that the password be exchanged on a secure channel, so as to prevent it from falling into the hands of an eavesdropper. A possible method to establish such a channel is Password Authenticated Connection Establishment (PACE), which itself is a password-based Diffie-Hellman key agreement protocol, specified in the form of a smart card protocol in [3]. Thus, in addition to the user's password, another password is needed if PACE is used, namely the one from which the PACE authentication key is derived.

A simple choice allowing to reduce the number of the passwords that the user has to manage would be to employ the same password both as key derivation password, verified implicitly by means of the PACE protocol, and as direct use password, verified explicitly by comparison. However, this approach has the following shortcomings:

- A usual countermeasure against trial-and-error attacks aimed at disclosing the user's password consists of blocking its use after a number of consecutive verification failures exceeding a given threshold. If the PACE authentication key is derived from the user's password, such key has to be blocked as well. Thus, an additional PACE authentication key would be needed for any user's operation not requiring to be preceded by the verification of the user's password, but only to be performed on a secure channel, such as the verification of a Personal Unblocking Code (PUC) by means of command RESET RETRY

COUNTER [4] to unblock the password. On the contrary, a single PACE authentication key is sufficient for all user's operations provided it is independent of the user's password, which leads to a simpler system.

- The user is typically allowed to change her password, e.g. by means of command CHANGE REFERENCE DATA [4]. If the PACE authentication key is derived from the user's password, such key has to be changed as well. This gives rise to additional functional requirements which can be nontrivial to meet, particularly in the case of a preexisting implementation having to be adapted. For instance, if the key itself is stored on the smart card rather than being derived at run time from the user's password, which improves performance and prevents side channel attacks, the update of the password and the key must be performed as an atomic operation to ensure their consistency. On the contrary, the PACE authentication key can remain unchanged provided it is independent of the user's password, which leads to a simpler system.

Therefore, a PACE password distinct from the user's password seems to be preferable. As the user's password is a secret known by the user only, the derivation of the PACE authentication key from the user's password would guarantee the secrecy of the key as well. If the PACE authentication key is rather derived from an independent password, then a new question arises: is this key required to be secret?

In order to find the answer, it is useful to schematize the protocol applying the informal notation used in [10]. If Generic Mapping is employed as mapping method (cf. [3]), the protocol takes the following form, where agents U and C stand for a given user and her own smart card, step Cn for the nth command APDU, and step Rn for the nth response APDU (for further information, cf. [3] and [4]).

R1. $C \to U : \{s\}_K$
C2. $U \to C : PK_{Map,PCD}$
R2. $C \to U : PK_{Map,IC}$
C3. $U \to C : PK_{DH,PCD}$
R3. $C \to U : PK_{DH,IC}$
C4. $U \to C : \{PK_{DH,IC}\}_{KS}$
R4. $C \to U : \{PK_{DH,PCD}\}_{KS}$
C5. $U \to C : \{\textit{User's password}\}_{KS}$
R5. $C \to U : \{\textit{Success code}\}_{KS}$

Being irrelevant for the security analysis of the protocol, the initial MANAGE SECURITY ENVIRONMENT: SET AT command/response pair, as well as the first GENERAL AUTHENTICATE command requesting nonce s, are not included in the scheme.

In the response to the first GENERAL AUTHENTICATE command (step R1), the card returns nonce s encrypted with the PACE authentication key K.

In the second GENERAL AUTHENTICATE command/response pair (steps C2 and R2), the user and the card exchange the respective ephemeral public keys $PK_{Map,PCD} = [SK_{Map,PCD}]G$ and $PK_{Map,IC} = [SK_{Map,IC}]G$, where G is the static cryptographic group generator (the notation used in [1] is applied). Then, both parties compute the ephemeral generator $G' = [s + SK_{Map,PCD} \times SK_{Map,IC}]G$.

In the third GENERAL AUTHENTICATE command/response pair (steps C3 and R3), the user and the card exchange another pair of ephemeral public keys $PK_{DH,PCD} = [SK_{DH,PCD}]G'$ and $PK_{DH,IC} = [SK_{DH,IC}]G'$, and then compute the shared secret $[SK_{DH,PCD} \times SK_{DH,IC}]G'$, from which session keys KS_{Enc} and KS_{MAC} are derived. In order to abstract from unnecessary details, the above scheme considers a single session key KS.

In the last GENERAL AUTHENTICATE command/response pair (steps C4 and R4), the user and the card exchange the respective authentication tokens, obtained by computing a Message Authentication Code (MAC) of the ephemeral public keys $PK_{DH,IC}$ and $PK_{DH,PCD}$ with session key KS_{MAC}. In order to abstract from unnecessary details, the above scheme represents these MACs as cryptograms generated using the single session key KS.

Finally, in steps C5 and R5, the user sends her password to the card on the secure messaging channel established by session keys KS_{Enc} and KS_{MAC}, e.g. via command VERIFY [4], and the card returns the success status word 0x9000 [4] over the same channel. In order to abstract from unnecessary details, the above scheme represents both messages as cryptograms generated using the single session key KS.

So, what if the PACE authentication key K were stolen by an attacker – henceforth called *spy* as done in [10]? In this case, even if the user's terminal were protected from attacks, the spy could get hold of the user's password by replacing the user's smart card with a fake one capable of performing a remote data transmission, so as to pull off a *grandmaster chess attack* [5]. In this way, the following scenario would occur, where agents F and S stand for the fake card and the spy.

R1. $F \to U : \{s\}_K$
C2. $U \to F : PK_{Map,PCD}$
R2. $F \to U : PK_{Map,IC}$

C3. $U \rightarrow F : PK_{DH,PCD}$
R3. $F \rightarrow U : PK_{DH,IC}$
C4. $U \rightarrow F : \{PK_{DH,IC}\}_{KS}$
R4. $F \rightarrow U : \{PK_{DH,PCD}\}_{KS}$
C5. $U \rightarrow F : \{\textit{User's password}\}_{KS}$
C5'. $F \rightarrow S : \textit{User's password}$

Since the spy has stored key K in its memory, the fake card can encrypt nonce s with K, so that it computes the same session keys as the user in step R3. As a result, the user receives a correct authentication token in step R4, and then agrees to send her password to the fake card in step C5. At this point, in order to accomplish the attack, the fake card has to do nothing but decrypt the user's password and send it to the spy on a remote communication channel, which is what happens in the final step C5'.

This argument demonstrates that the answer to the pending question is affirmative, namely the PACE authentication key is indeed required to be secret, if Generic Mapping is used. Moreover, the same conclusion can be drawn on the basis of a similar argument in case the mapping method being used is Integrated Mapping (cf. [3]). Therefore, the PACE password from which the key is derived must be secret as well.

This requirement has a significant impact on both the security and the usability of the system. In fact, the only way to prevent the user from having to input the PACE password in addition to the direct use one is providing such password to the user's terminal by other means. In the case of a stand-alone application, this implies that either the PACE password itself or data allowing its computation must be stored somewhere in the user's terminal, which gives rise to a risk of leakage. The alternative is to have the PACE password typed in by the user, which renders longer the overall credentials that the user is in charge of managing securely. Furthermore, any operation having to be performed on a secure messaging channel before the user types in her password – such as identifying the user in case the smart card is endowed with an identity application compliant with [2] and [3] – would require an additional PACE password independent of the user's one. Hence, such preliminary operations and the subsequent user's password verification would have to be performed on distinct secure messaging channels, which would cause a deterioration in the system performance.

In case Chip Authentication Mapping is used as mapping method instead (cf. [3]), the resulting protocol can be schematized as follows.

R1. $C \rightarrow U : \{s\}_K$
C2. $U \rightarrow C : PK_{Map,PCD}$

R2. $C \rightarrow U : PK_{Map,IC}$
C3. $U \rightarrow C : PK_{DH,PCD}$
R3. $C \rightarrow U : PK_{DH,IC}$
C4. $U \rightarrow C : \{PK_{DH,IC}\}_{KS}$
R4. $C \rightarrow U : \{PK_{DH,PCD}, (SK_{IC})^{-1} \times SK_{Map,IC} \ mod \ n,$
$PK_{IC}, PK_{IC} \ signature\}_{KS}$
C5. $U \rightarrow C : \{User's \ password\}_{KS}$
R5. $C \rightarrow U : \{Success \ code\}_{KS}$

In the response to the last GENERAL AUTHENTICATE command (step R4), in addition to the MAC of $PK_{DH,PCD}$ computed with session key KS_{MAC}, the smart card returns also the *Encrypted Chip Authentication Data* (A_{IC}) if Chip Authentication Mapping is used. These data result from the encryption with session key KS_{Enc} of the *Chip Authentication Data* (CA_{IC}), which consist of the product modulo n, where n is the group order, of the inverse modulo n of the static private key SK_{IC} with the ephemeral private key $SK_{Map,IC}$.
The user can then verify the authenticity of the chip applying the following procedure.

1. Read the static public key $PK_{IC} = [SK_{IC}]G$ from a dedicated file of the smart card, named *EF.CardSecurity*.
   Because of the read access conditions to be enforced by this file, it must be read over the secure messaging channel established by session keys KS_{Enc} and KS_{MAC} (cf. [2]).

2. Verify the signature contained in file EF.CardSecurity, generated over the contents of the file by a trusted Certification Authority (CA).
   To perform this operation, the user's terminal is supposed to be provided by secure means with the public key corresponding to the private key used by the CA for signature generation.

3. Decrypt the received A_{IC} to recover CA_{IC} and verify that $[CA_{IC}]PK_{IC} = PK_{Map,IC}$.
   Since this happens just in case $CA_{IC} = (SK_{IC})^{-1} \times SK_{Map,IC} \ mod \ n$, the success of such verification proves that the chip knows the private key SK_{IC} corresponding to the certified public key PK_{IC}, and thus is authentic.

The reading of file EF.CardSecurity is performed next to the last GENERAL AUTHENTICATE command as a separate operation, by sending one or more READ BINARY commands on the secure messaging channel established by session keys KS_{Enc} and KS_{MAC} (cf. [2], [3], and [4]). The

above scheme represents this operation by inserting the public key PK_{IC} and its signature into the cryptogram returned by the last GENERAL AUTHENTICATE command, so as to abstract from unnecessary details once again.

A successful verification of Chip Authentication Data provides the user with a proof of the fact that the party knowing private key $SK_{Map,IC}$, and then sharing the same session keys KS_{Enc} and KS_{MAC}, is an authentic chip. Thus, the protocol ensures that the user accepts to send her password to an authentic chip only. As a result, the grandmaster chess attack described previously is not applicable, so that the user's password cannot be stolen by the spy any longer. What is more, this is true independently of the secrecy of the PACE authentication key. Therefore, this key is no longer required to be secret, which solves all the problems ensuing from such requirement.

The purpose of this paper is indeed to construct a formal model of the above protocol in the Chip Authentication Mapping case and prove its security, applying Paulson's Inductive Method as described in [10]. In more detail, the formal development is aimed at proving that such protocol enforces the following security properties.

- Secrecy theorem *pr-key-secrecy*: if a user other than the spy sends her password to some smart card (not necessarily her own one), then the spy cannot disclose the session key used to encrypt the password. This property ensures that the protocol is successful in establishing trustworthy secure messaging channels between users and smart cards.
- Secrecy theorem *pr-passwd-secrecy*: the spy cannot disclose the passwords of other users. This property ensures that the protocol is successful in preserving the secrecy of users' passwords.
- Authenticity theorem *pr-user-authenticity*: if a smart card receives the password of a user (not necessarily the cardholder), then the message must have been originally sent by that user. This property ensures that the protocol enables users to authenticate themselves to their smart cards, viz. provides an *external authentication* service (cf. [4]).
- Authenticity theorem *pr-card-authenticity*: if a user sends her password to a smart card and receives a success code as response, then the card is her own one and the response must have been originally sent by that card. This property ensures that the protocol enables smart cards to authenticate themselves to their cardholders, viz. provides an *internal authentication* service (cf. [4]).

Remarkably, none of these theorems turns out to require the secrecy of the PACE authentication key as an assumption, so that all of them are valid independently of whether this key is secret or not.

The main technical difficulties arising from this formal development are the following ones.

- Data such as private keys for Diffie-Hellman key agreement and session keys do not necessarily occur as components of exchanged messages, viz. they may be computed by some agent without being ever sent to any other agent. In this case, whichever protocol trace *evs* is given, any such key x will not be contained in either set *analz* (*spies evs*) or *used evs*, so that statements such as $x \in$ *analz* (*spies evs*) or $x \in$ *used evs* will be vacuously false. Thus, some way must be found to formalize a state of affairs where x is known by the spy or has already been used in some protocol run.

- As private keys for Diffie-Hellman key agreement do not necessarily occur as components of exchanged messages, some way must be found to record the private keys that each agent has either generated or accepted from some other agent (possibly implicitly, in the form of the corresponding public keys) in each protocol run.

- The public keys for Diffie-Hellman key agreement being used are comprised of the elements of a cryptographic cyclic group of prime order n, and the private keys are the elements of the finite field comprised of the integers from 0 to n - 1 (cf. [3], [1]). Hence, the operations defined in these algebraic structures, as well as the generation of public keys from known private keys, correspond to additional ways in which the spy can generate fake messages starting from known ones. A possible option to reflect this in the formal model would be to extend the inductive definition of set *synth H* with rules enabling to obtain new Diffie-Hellman private and public keys from those contained in set H, but the result would be an overly complex definition. Thus, an alternative formalization ought to be found.

These difficulties are solved by extending the Inductive Method, with respect to the form specified in [10], as follows.

- The protocol is no longer defined as a set of event lists, but rather as a set of 4-tuples (*evs*, S, A, U) where *evs* is an event list, S is the current protocol *state* – viz. a function that maps each agent to the private keys for Diffie-Hellman key agreement generated or accepted in each protocol run –, A is the set of the Diffie-Hellman private keys and session keys currently known by the spy, and U is the set of the Diffie-Hellman private keys and session keys which have already been used in some protocol run.
  In this way, the first two difficulties are solved. Particularly, the full

set of the messages currently known by the spy can be formalized as the set *analz* ($A \cup$ *spies evs*).

- The inductive definition of the protocol does not contain a single *fake* rule any longer, but rather one *fake* rule for each protocol step. Each *fake* rule is denoted by adding letter "F" to the identifier of the corresponding protocol step, e.g. the *fake* rules associated to steps C2 and R5 are given the names *FC2* and *FR5*, respectively.
  In this way, the third difficulty is solved, too. In fact, for each protocol step, the related *fake* rule extends the spy's capabilities to generate fake messages with the operations on known Diffie-Hellman private and public keys relevant for that step, which makes an augmentation of set *synth H* with such operations unnecessary.

Throughout this paper, the salient points of definitions and proofs are commented; for additional information, cf. Isabelle documentation, particularly [9], [8], [7], and [6].
Paulson's Inductive Method is described in [10], and further information is provided in [9] as a case study. The formal developments described in [10] and [9] are included in the Isabelle distribution.
Additional information on the involved cryptography can be found in [3] and [1].

1.2 Propaedeutic definitions

First of all, the data types of encryption/signature keys, Diffie-Hellman private keys, and Diffie-Hellman public keys are defined. Following [9], encryption/signature keys are identified with natural numbers, whereas Diffie-Hellman private keys and public keys are represented as rational and integer numbers in order to model the algebraic structures that they form (a field and a group, respectively; cf. above).

type-synonym *key* = *nat*

type-synonym *pri-agrk* = *rat*

type-synonym *pub-agrk* = *int*

Agents are comprised of an infinite quantity of users and smart cards, plus the Certification Authority (CA) signing public key PK_{IC}. For each n, *User* n is the cardholder of smart card *Card* n.

datatype *agent* = *CA* | *Card nat* | *User nat*

In addition to the kinds of messages considered in [10], the data type of messages comprises also users' passwords, Diffie-Hellman private and public keys, and Chip Authentication Data. Particularly, for each n, *Passwd* n is the password of *User* n, accepted as being the correct one by *Card* n.

datatype *msg* =
  Agent *agent* |
  Number *nat* |
  Nonce *nat* |
  Key *key* |
  Hash *msg* |
  Passwd *nat* |
  Pri-AgrK *pri-agrk* |
  Pub-AgrK *pub-agrk* |
  Auth-Data *pri-agrk pri-agrk* |
  Crypt *key msg* |
  MPair *msg msg*

syntax
  -*MTuple* :: [$'a$, *args*] $\Rightarrow$ $'a * 'b$ ((2{|-,/ -|}))

translations
  {|x, y, z|} $\rightleftharpoons$ {|x, {|y, z|}|}
  {|x, y|} $\rightleftharpoons$ *CONST MPair* x y

As regards data type *event*, constructor *Says* is extended with three additional parameters of type *nat*, respectively identifying the communication channel, the protocol run, and the protocol step (ranging from 1 to 5) in which the message is exchanged. Communication channels are associated to smart cards, so that if a user receives an encrypted nonce s on channel n, she will answer by sending her ephemeral public key $PK_{Map,PCD}$ for generator mapping to smart card *Card* n.

datatype *event* = *Says nat nat nat agent agent msg*

The record data type *session* is used to store the Diffie-Hellman private keys that each agent has generated or accepted in each protocol run. In more detail:

- Field *NonceS* is deputed to contain the nonce s, if any, having been generated internally (in the case of a smart card) or accepted from the external world (in the case of a user).

- Field *IntMapK* is deputed to contain the ephemeral private key for generator mapping, if any, having been generated internally.
- Field *ExtMapK* is deputed to contain the ephemeral private key for generator mapping, if any, having been implicitly accepted from the external world in the form of the corresponding public key.
- Field *IntAgrK* is deputed to contain the ephemeral private key for key agreement, if any, having been generated internally.
- Field *ExtAgrK* is deputed to contain the ephemeral private key for key agreement, if any, having been implicitly accepted from the external world in the form of the corresponding public key.

record *session* =
  NonceS :: *pri-agrk option*
  IntMapK :: *pri-agrk option*
  ExtMapK :: *pri-agrk option*
  IntAgrK :: *pri-agrk option*
  ExtAgrK :: *pri-agrk option*

Then, the data type of protocol states is defined as the type of the functions that map any 3-tuple (X, n, *run*), where X is an agent, n identifies a communication channel, and *run* identifies a protocol run taking place on that communication channel, to a record of type *session*.

type-synonym *state* = *agent* × *nat* × *nat* ⇒ *session*

Set *bad* collects the numerical identifiers of the PACE authentication keys known by the spy, viz. for each n, $n \in$ *bad* just in case the spy knows the PACE authentication key shared by agents *User n* and *Card n*.

consts *bad* :: *nat set*

Function *invK* maps each encryption/signature key to the corresponding inverse key, matching the original key just in case it is symmetric.

consts *invK* :: *key* ⇒ *key*

Function *agrK* maps each Diffie-Hellman private key x to the corresponding public key $[x]G$, where G is the static cryptographic group generator being used.

consts *agrK* :: *pri-agrk* ⇒ *pub-agrk*

Function *sesK* maps each Diffie-Hellman private key x to the session key resulting from shared secret $[x]G$, where G is the static cryptographic group generator being used.

consts *sesK* :: *pri-agrk* ⇒ *key*

Function *symK* maps each natural number n to the PACE authentication key shared by agents *User n* and *Card n*.

consts *symK* :: *nat* ⇒ *key*

Function *priAK* maps each natural number n to the static Diffie-Hellman private key SK_{IC} assigned to smart card *Card n* for Chip Authentication.

consts *priAK* :: *nat* ⇒ *pri-agrk*

Function *priSK* maps each agent to her own private key for digital signature generation, even if the only such key being actually significant for the model is the Certification Authority's one, i.e. *priSK CA*.

consts *priSK* :: *agent* ⇒ *key*

The spy is modeled as a user, specifically the one identified by number 0, i.e. *User 0*. In this way, in addition to the peculiar privilege of being able to generate fake messages, the spy is endowed with the capability of performing any operation that a generic user can do.

abbreviation *Spy* :: *agent* **where**
Spy ≡ *User 0*

Functions *pubAK* and *pubSK* are abbreviations useful to make the formal development more readable. The former function maps each Diffie-Hellman private key x to the message comprised of the corresponding public key *agrK* x, whereas the latter maps each agent to the corresponding public key for digital signature verification.

abbreviation *pubAK* :: *pri-agrk* ⇒ *msg* **where**
pubAK a ≡ *Pub-AgrK* (*agrK a*)

abbreviation *pubSK* :: *agent* ⇒ *key* **where**
pubSK X ≡ *invK* (*priSK X*)

Function *start-S* represents the initial protocol state, i.e. the one in which no ephemeral Diffie-Hellman private key has been generated or accepted by any agent yet.

abbreviation *start-S* :: *state* **where**
start-S ≡ λ*x*. (|*NonceS* = *None*, *IntMapK* = *None*, *ExtMapK* = *None*,
IntAgrK = *None*, *ExtAgrK* = *None*|)

Set *start-A* is comprised of the messages initially known by the spy, namely:

- her own password as a user,
- the compromised PACE authentication keys,
- the public keys for digital signature verification, and
- the static Diffie-Hellman public keys assigned to smart cards for Chip Authentication.

abbreviation *start-A* :: *msg set* **where**
start-A ≡ *insert* (*Passwd 0*) (*Key ‘ symK ‘ bad* ∪ *Key ‘ range pubSK* ∪ *pubAK ‘ range priAK*)

Set *start-U* is comprised of the messages which have already been used before the execution of the protocol starts, namely:

- all users' passwords,
- all PACE authentication keys,
- the private and public keys for digital signature generation/verification, and
- the static Diffie-Hellman private and public keys assigned to smart cards for Chip Authentication.

abbreviation *start-U* :: *msg set* **where**
start-U ≡ *range Passwd* ∪ *Key* ‘ *range symK* ∪ *Key* ‘ *range priSK* ∪ *Key* ‘ *range pubSK* ∪
Pri-AgrK ‘ *range priAK* ∪ *pubAK* ‘ *range priAK*

As in [10], function *spies* models the set of the messages that the spy can see in a protocol trace. However, it is no longer necessary to identify *spies* [] with the initial knowledge of the spy, since her current knowledge in correspondence with protocol state (*evs*, *S*, *A*, *U*) is represented as set *analz* (*A* ∪ *spies evs*), where *start-A* ⊆ *A*. Therefore, this formal development defines *spies* [] as the empty set.

fun *spies* :: *event list* ⇒ *msg set* **where**
spies [] = {} |
spies (*Says i j k A B X* # *evs*) = *insert X* (*spies evs*)

Here below is the specification of the axioms about the constants defined previously which are used in the formal proofs. A model of the constants satisfying the axioms is also provided in order to ensure the consistency of the formal development. In more detail:

1. Axiom *agrK-inj* states that function *agrK* is injective, and formalizes the fact that distinct Diffie-Hellman private keys generate distinct public keys.
   Since the former keys are represented as rational numbers and the latter as integer numbers (cf. above), a model of function *agrK* satisfying the axiom is built by means of the injective function *inv nat-to-rat-surj* provided by the Isabelle distribution, which maps rational numbers to natural numbers.

2. Axiom *sesK-inj* states that function *sesK* is injective, and formalizes the fact that the key derivation function specified in [3] for deriving session keys from shared secrets makes use of robust hash functions, so that collisions are negligible.
   Since Diffie-Hellman private keys are represented as rational numbers and encryption/signature keys as natural numbers (cf. above), a model of function *sesK* satisfying the axiom is built by means of the injective function *inv nat-to-rat-surj*, too.

3. Axiom *priSK-pubSK* formalizes the fact that every private key for signature generation is distinct from whichever public key for signature verification. For example, in the case of the RSA algorithm, small fixed

values are typically used as public exponents to make signature verification more efficient, whereas the corresponding private exponents are of the same order of magnitude as the modulus.

4. Axiom *priSK-symK* formalizes the fact that private keys for signature generation are distinct from PACE authentication keys, which is obviously true since the former keys are asymmetric whereas the latter are symmetric.

5. Axiom *pubSK-symK* formalizes the fact that public keys for signature verification are distinct from PACE authentication keys, which is obviously true since the former keys are asymmetric whereas the latter are symmetric.

6. Axiom *invK-sesK* formalizes the fact that session keys are symmetric.

7. Axiom *invK-symK* formalizes the fact that PACE authentication keys are symmetric.

8. Axiom *symK-bad* states that set *bad* is closed with respect to the identity of PACE authentication keys, viz. if a compromised user has the same PACE authentication key as another user, then the latter user is compromised as well.

It is worth remarking that there is no axiom stating that distinct PACE authentication keys are assigned to distinct users. As a result, the formal development does not depend on the enforcement of this condition.

specification (*bad invK agrK sesK symK priSK*)
agrK-inj: $inj\ agrK$
sesK-inj: $inj\ sesK$
priSK-pubSK: $priSK\ X \neq pubSK\ X'$
priSK-symK: $priSK\ X \neq symK\ n$
pubSK-symK: $pubSK\ X \neq symK\ n$
invK-sesK: $invK\ (sesK\ a) = sesK\ a$
invK-symK: $invK\ (symK\ n) = symK\ n$
symK-bad: $m \in bad \Longrightarrow symK\ n = symK\ m \Longrightarrow n \in bad$
⟨*proof*⟩

Here below are the inductive definitions of sets *parts*, *analz*, and *synth*. With respect to the definitions given in the protocol library included in the Isabelle distribution, those of *parts* and *analz* are extended with rules extracting Diffie-Hellman private keys from Chip Authentication Data, whereas the definition of *synth* contains a further rule that models the inverse operation, i.e. the construction of Chip Authentication Data starting from private keys. Particularly, the additional *analz* rules formalize the fact that, for any two

private keys x and y, if $x \times y \ mod \ n$ and x are known, where n is the group order, then y can be obtained by computing $x \times y \times x^{-1} \ mod \ n$, and similarly, x can be obtained if y is known.

An additional set, named *items*, is also defined inductively in what follows. This set is a hybrid of *parts* and *analz*, as it shares with *parts* the rule applying to cryptograms and with *analz* the rules applying to Chip Authentication Data. Since the former rule is less strict than the corresponding one in the definition of *analz*, it turns out that $analz\ H \subseteq items\ H$ for any message set H. As a result, for any message X, $X \notin items\ (A \cup spies\ evs)$ implies $X \notin analz\ (A \cup spies\ evs)$. Therefore, set *items* is useful to prove the secrecy of the Diffie-Hellman private keys utilized to compute Chip Authentication Data without bothering with case distinctions concerning the secrecy of encryption keys, as would happen if set *analz* were directly employed instead.

inductive-set *parts* :: *msg set* ⇒ *msg set* **for** H :: *msg set* **where**
Inj: $X \in H \Longrightarrow X \in parts\ H$ |
Fst: $\{\!|X, Y|\!\} \in parts\ H \Longrightarrow X \in parts\ H$ |
Snd: $\{\!|X, Y|\!\} \in parts\ H \Longrightarrow Y \in parts\ H$ |
Body: $Crypt\ K\ X \in parts\ H \Longrightarrow X \in parts\ H$ |
Auth-Fst: $Auth\text{-}Data\ x\ y \in parts\ H \Longrightarrow Pri\text{-}AgrK\ x \in parts\ H$ |
Auth-Snd: $Auth\text{-}Data\ x\ y \in parts\ H \Longrightarrow Pri\text{-}AgrK\ y \in parts\ H$

inductive-set *items* :: *msg set* ⇒ *msg set* **for** H :: *msg set* **where**
Inj: $X \in H \Longrightarrow X \in items\ H$ |
Fst: $\{\!|X, Y|\!\} \in items\ H \Longrightarrow X \in items\ H$ |
Snd: $\{\!|X, Y|\!\} \in items\ H \Longrightarrow Y \in items\ H$ |
Body: $Crypt\ K\ X \in items\ H \Longrightarrow X \in items\ H$ |
Auth-Fst: $[\![Auth\text{-}Data\ x\ y \in items\ H;\ Pri\text{-}AgrK\ y \in items\ H]\!] \Longrightarrow Pri\text{-}AgrK\ x \in items\ H$ |
Auth-Snd: $[\![Auth\text{-}Data\ x\ y \in items\ H;\ Pri\text{-}AgrK\ x \in items\ H]\!] \Longrightarrow Pri\text{-}AgrK\ y \in items\ H$

inductive-set *analz* :: *msg set* ⇒ *msg set* **for** H :: *msg set* **where**
Inj: $X \in H \Longrightarrow X \in analz\ H$ |
Fst: $\{\!|X, Y|\!\} \in analz\ H \Longrightarrow X \in analz\ H$ |
Snd: $\{\!|X, Y|\!\} \in analz\ H \Longrightarrow Y \in analz\ H$ |
Decrypt: $[\![Crypt\ K\ X \in analz\ H;\ Key\ (invK\ K) \in analz\ H]\!] \Longrightarrow X \in analz\ H$ |
Auth-Fst: $[\![Auth\text{-}Data\ x\ y \in analz\ H;\ Pri\text{-}AgrK\ y \in analz\ H]\!] \Longrightarrow Pri\text{-}AgrK\ x \in analz\ H$ |
Auth-Snd: $[\![Auth\text{-}Data\ x\ y \in analz\ H;\ Pri\text{-}AgrK\ x \in analz\ H]\!] \Longrightarrow Pri\text{-}AgrK\ y \in analz\ H$

inductive-set *synth* :: *msg set* ⇒ *msg set* **for** H :: *msg set* **where**
Inj: $X \in H \Longrightarrow X \in synth\ H$ |
Agent: $Agent\ X \in synth\ H$ |
Number: $Number\ n \in synth\ H$ |
Hash: $X \in synth\ H \Longrightarrow Hash\ X \in synth\ H$ |

MPair: $[\![X \in synth\ H;\ Y \in synth\ H]\!] \Longrightarrow \{\!|X,\ Y|\!\} \in synth\ H\ |$
Crypt: $[\![X \in synth\ H;\ Key\ K \in H]\!] \Longrightarrow Crypt\ K\ X \in synth\ H\ |$
Auth: $[\![Pri\text{-}AgrK\ x \in H;\ Pri\text{-}AgrK\ y \in H]\!] \Longrightarrow Auth\text{-}Data\ x\ y \in synth\ H$

1.3 Propaedeutic lemmas

This section contains the lemmas about sets *parts*, *items*, *analz*, and *synth* required for protocol verification. Since their proofs mainly consist of initial rule inductions followed by sequences of rule applications and simplifications, *apply*-style is used.

lemma *set-spies* [*rule-format*]:
$Says\ i\ j\ k\ A\ B\ X \in set\ evs \longrightarrow X \in spies\ evs$
⟨*proof*⟩

lemma *parts-subset*:
$H \subseteq parts\ H$
⟨*proof*⟩

lemma *parts-idem*:
$parts\ (parts\ H) = parts\ H$
⟨*proof*⟩

lemma *parts-simp*:
$H \subseteq range\ Agent \cup$
$range\ Number \cup$
$range\ Nonce \cup$
$range\ Key \cup$
$range\ Hash \cup$
$range\ Passwd \cup$
$range\ Pri\text{-}AgrK \cup$
$range\ Pub\text{-}AgrK \Longrightarrow$
$parts\ H = H$
⟨*proof*⟩

lemma *parts-mono*:
$G \subseteq H \Longrightarrow parts\ G \subseteq parts\ H$
⟨*proof*⟩

lemma *parts-insert*:
$insert\ X\ (parts\ H) \subseteq parts\ (insert\ X\ H)$
⟨*proof*⟩

lemma *parts-simp-insert*:
$X \in range\ Agent \cup$
$range\ Number \cup$
$range\ Nonce \cup$
$range\ Key \cup$

$range\ Hash \cup$
$range\ Passwd \cup$
$range\ Pri\text{-}AgrK \cup$
$range\ Pub\text{-}AgrK \Longrightarrow$
$parts\ (insert\ X\ H) = insert\ X\ (parts\ H)$
⟨*proof*⟩

lemma *parts-auth-data-1*:
$parts\ (insert\ (Auth\text{-}Data\ x\ y)\ H) \subseteq$
$\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y,\ Auth\text{-}Data\ x\ y\} \cup parts\ H$
⟨*proof*⟩

lemma *parts-auth-data-2*:
$\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y,\ Auth\text{-}Data\ x\ y\} \cup parts\ H \subseteq$
$parts\ (insert\ (Auth\text{-}Data\ x\ y)\ H)$
⟨*proof*⟩

lemma *parts-auth-data*:
$parts\ (insert\ (Auth\text{-}Data\ x\ y)\ H) =$
$\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y,\ Auth\text{-}Data\ x\ y\} \cup parts\ H$
⟨*proof*⟩

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

lemma *parts-crypt-2*:
$insert\ (Crypt\ K\ X)\ (parts\ (insert\ X\ H)) \subseteq parts\ (insert\ (Crypt\ K\ X)\ H)$
⟨*proof*⟩

lemma *parts-crypt*:
$parts\ (insert\ (Crypt\ K\ X)\ H) = insert\ (Crypt\ K\ X)\ (parts\ (insert\ X\ H))$
⟨*proof*⟩

lemma *parts-mpair-1*:
$parts\ (insert\ \{\!|X,\ Y|\!\}\ H) \subseteq insert\ \{\!|X,\ Y|\!\}\ (parts\ (\{X,\ Y\} \cup H))$
⟨*proof*⟩

lemma *parts-mpair-2*:
$insert\ \{\!|X,\ Y|\!\}\ (parts\ (\{X,\ Y\} \cup H)) \subseteq parts\ (insert\ \{\!|X,\ Y|\!\}\ H)$
⟨*proof*⟩

lemma *parts-mpair*:
$parts\ (insert\ \{\!|X,\ Y|\!\}\ H) = insert\ \{\!|X,\ Y|\!\}\ (parts\ (\{X,\ Y\} \cup H))$
⟨*proof*⟩

lemma *items-subset*:
$H \subseteq items\ H$
⟨*proof*⟩

lemma *items-idem*:
$items\ (items\ H) = items\ H$
⟨*proof*⟩

lemma *items-parts-subset*:
$items\ H \subseteq parts\ H$
⟨*proof*⟩

lemma *items-simp*:
$H \subseteq range\ Agent \cup$
$range\ Number \cup$
$range\ Nonce \cup$
$range\ Key \cup$
$range\ Hash \cup$
$range\ Passwd \cup$
$range\ Pri\text{-}AgrK \cup$
$range\ Pub\text{-}AgrK \Longrightarrow$
$items\ H = H$
⟨*proof*⟩

lemma *items-mono*:
$G \subseteq H \Longrightarrow items\ G \subseteq items\ H$
⟨*proof*⟩

lemma *items-insert*:
$insert\ X\ (items\ H) \subseteq items\ (insert\ X\ H)$
⟨*proof*⟩

lemma *items-simp-insert-1*:
$X \in items\ H \Longrightarrow items\ (insert\ X\ H) = items\ H$
⟨*proof*⟩

lemma *items-simp-insert-2*:
$X \in range\ Agent \cup$
$range\ Number \cup$
$range\ Nonce \cup$
$range\ Key \cup$
$range\ Hash \cup$
$range\ Passwd \cup$
$range\ Pub\text{-}AgrK \Longrightarrow$
$items\ (insert\ X\ H) = insert\ X\ (items\ H)$
⟨*proof*⟩

lemma *items-pri-agrk-out*:
$Pri\text{-}AgrK\ x \notin parts\ H \Longrightarrow$
$items\ (insert\ (Pri\text{-}AgrK\ x)\ H) = insert\ (Pri\text{-}AgrK\ x)\ (items\ H)$
⟨*proof*⟩

lemma *items-auth-data-in-1*:
$items\ (insert\ (Auth\text{-}Data\ x\ y)\ H) \subseteq$
$insert\ (Auth\text{-}Data\ x\ y)\ (items\ (\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y\} \cup H))$
⟨*proof*⟩

lemma *items-auth-data-in-2*:
$Pri\text{-}AgrK\ x \in items\ H \vee Pri\text{-}AgrK\ y \in items\ H \Longrightarrow$
$insert\ (Auth\text{-}Data\ x\ y)\ (items\ (\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y\} \cup H)) \subseteq$
$items\ (insert\ (Auth\text{-}Data\ x\ y)\ H)$
⟨*proof*⟩

lemma *items-auth-data-in*:
$Pri\text{-}AgrK\ x \in items\ H \vee Pri\text{-}AgrK\ y \in items\ H \Longrightarrow$
$items\ (insert\ (Auth\text{-}Data\ x\ y)\ H) =$
$insert\ (Auth\text{-}Data\ x\ y)\ (items\ (\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y\} \cup H))$
⟨*proof*⟩

lemma *items-auth-data-out*:
$[\![Pri\text{-}AgrK\ x \notin items\ H;\ Pri\text{-}AgrK\ y \notin items\ H]\!] \Longrightarrow$
$items\ (insert\ (Auth\text{-}Data\ x\ y)\ H) = insert\ (Auth\text{-}Data\ x\ y)\ (items\ H)$
⟨*proof*⟩

lemma *items-crypt-1*:
$items\ (insert\ (Crypt\ K\ X)\ H) \subseteq insert\ (Crypt\ K\ X)\ (items\ (insert\ X\ H))$
⟨*proof*⟩

lemma *items-crypt-2*:
$insert\ (Crypt\ K\ X)\ (items\ (insert\ X\ H)) \subseteq items\ (insert\ (Crypt\ K\ X)\ H)$
⟨*proof*⟩

lemma *items-crypt*:
$items\ (insert\ (Crypt\ K\ X)\ H) = insert\ (Crypt\ K\ X)\ (items\ (insert\ X\ H))$
⟨*proof*⟩

lemma *items-mpair-1*:
$items\ (insert\ \{\!|X,\ Y|\!\}\ H) \subseteq insert\ \{\!|X,\ Y|\!\}\ (items\ (\{X,\ Y\} \cup H))$
⟨*proof*⟩

lemma *items-mpair-2*:
$insert\ \{\!|X,\ Y|\!\}\ (items\ (\{X,\ Y\} \cup H)) \subseteq items\ (insert\ \{\!|X,\ Y|\!\}\ H)$
⟨*proof*⟩

lemma *items-mpair*:
$items\ (insert\ \{\!|X,\ Y|\!\}\ H) = insert\ \{\!|X,\ Y|\!\}\ (items\ (\{X,\ Y\} \cup H))$
⟨*proof*⟩

lemma *analz-subset*:
$H \subseteq analz\ H$
⟨*proof*⟩

lemma *analz-idem*:
$analz\ (analz\ H) = analz\ H$
⟨*proof*⟩

lemma *analz-parts-subset*:
$analz\ H \subseteq parts\ H$
⟨*proof*⟩

lemma *analz-items-subset*:
$analz\ H \subseteq items\ H$
⟨*proof*⟩

lemma *analz-simp*:
$H \subseteq range\ Agent \cup$
$range\ Number \cup$
$range\ Nonce \cup$
$range\ Key \cup$
$range\ Hash \cup$
$range\ Passwd \cup$
$range\ Pri\text{-}AgrK \cup$
$range\ Pub\text{-}AgrK \Longrightarrow$
$analz\ H = H$
⟨*proof*⟩

lemma *analz-mono*:
$G \subseteq H \Longrightarrow analz\ G \subseteq analz\ H$
⟨*proof*⟩

lemma *analz-insert*:
$insert\ X\ (analz\ H) \subseteq analz\ (insert\ X\ H)$
⟨*proof*⟩

lemma *analz-simp-insert-1*:
$X \in analz\ H \Longrightarrow analz\ (insert\ X\ H) = analz\ H$
⟨*proof*⟩

lemma *analz-simp-insert-2*:
$X \in range\ Agent \cup$
$range\ Number \cup$
$range\ Nonce \cup$
$range\ Hash \cup$
$range\ Passwd \cup$
$range\ Pub\text{-}AgrK \Longrightarrow$
$analz\ (insert\ X\ H) = insert\ X\ (analz\ H)$
⟨*proof*⟩

lemma *analz-auth-data-in-1*:
$analz\ (insert\ (Auth\text{-}Data\ x\ y)\ H) \subseteq$

$insert\ (Auth\text{-}Data\ x\ y)\ (analz\ (\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y\} \cup H))$
⟨*proof*⟩

lemma *analz-auth-data-in-2*:
$Pri\text{-}AgrK\ x \in analz\ H \vee Pri\text{-}AgrK\ y \in analz\ H \Longrightarrow$
$insert\ (Auth\text{-}Data\ x\ y)\ (analz\ (\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y\} \cup H)) \subseteq$
$analz\ (insert\ (Auth\text{-}Data\ x\ y)\ H)$
⟨*proof*⟩

lemma *analz-auth-data-in*:
$Pri\text{-}AgrK\ x \in analz\ H \vee Pri\text{-}AgrK\ y \in analz\ H \Longrightarrow$
$analz\ (insert\ (Auth\text{-}Data\ x\ y)\ H) =$
$insert\ (Auth\text{-}Data\ x\ y)\ (analz\ (\{Pri\text{-}AgrK\ x,\ Pri\text{-}AgrK\ y\} \cup H))$
⟨*proof*⟩

lemma *analz-auth-data-out*:
$[\![Pri\text{-}AgrK\ x \notin analz\ H;\ Pri\text{-}AgrK\ y \notin analz\ H]\!] \Longrightarrow$
$analz\ (insert\ (Auth\text{-}Data\ x\ y)\ H) = insert\ (Auth\text{-}Data\ x\ y)\ (analz\ H)$
⟨*proof*⟩

lemma *analz-crypt-in-1*:
$analz\ (insert\ (Crypt\ K\ X)\ H) \subseteq insert\ (Crypt\ K\ X)\ (analz\ (insert\ X\ H))$
⟨*proof*⟩

lemma *analz-crypt-in-2*:
$Key\ (invK\ K) \in analz\ H \Longrightarrow$
$insert\ (Crypt\ K\ X)\ (analz\ (insert\ X\ H)) \subseteq analz\ (insert\ (Crypt\ K\ X)\ H)$
⟨*proof*⟩

lemma *analz-crypt-in*:
$Key\ (invK\ K) \in analz\ H \Longrightarrow$
$analz\ (insert\ (Crypt\ K\ X)\ H) = insert\ (Crypt\ K\ X)\ (analz\ (insert\ X\ H))$
⟨*proof*⟩

lemma *analz-crypt-out*:
$Key\ (invK\ K) \notin analz\ H \Longrightarrow$
$analz\ (insert\ (Crypt\ K\ X)\ H) = insert\ (Crypt\ K\ X)\ (analz\ H)$
⟨*proof*⟩

lemma *analz-mpair-1*:
$analz\ (insert\ \{\!|X,\ Y|\!\}\ H) \subseteq insert\ \{\!|X,\ Y|\!\}\ (analz\ (\{X,\ Y\} \cup H))$
⟨*proof*⟩

lemma *analz-mpair-2*:
$insert\ \{\!|X,\ Y|\!\}\ (analz\ (\{X,\ Y\} \cup H)) \subseteq analz\ (insert\ \{\!|X,\ Y|\!\}\ H)$
⟨*proof*⟩

lemma *analz-mpair*:
$analz\ (insert\ \{\!|X,\ Y|\!\}\ H) = insert\ \{\!|X,\ Y|\!\}\ (analz\ (\{X,\ Y\} \cup H))$

⟨*proof*⟩

lemma *synth-simp-intro*:
$X \in synth\ H \Longrightarrow$
$X \in range\ Nonce \cup$
$range\ Key \cup$
$range\ Passwd \cup$
$range\ Pri\text{-}AgrK \cup$
$range\ Pub\text{-}AgrK \Longrightarrow$
$X \in H$
⟨*proof*⟩

lemma *synth-auth-data*:
$Auth\text{-}Data\ x\ y \in synth\ H \Longrightarrow$
$Auth\text{-}Data\ x\ y \in H \vee Pri\text{-}AgrK\ x \in H \wedge Pri\text{-}AgrK\ y \in H$
⟨*proof*⟩

lemma *synth-crypt*:
$Crypt\ K\ X \in synth\ H \Longrightarrow Crypt\ K\ X \in H \vee X \in synth\ H \wedge Key\ K \in H$
⟨*proof*⟩

lemma *synth-mpair*:
$\{\!|X,\ Y|\!\} \in synth\ H \Longrightarrow \{\!|X,\ Y|\!\} \in H \vee X \in synth\ H \wedge Y \in synth\ H$
⟨*proof*⟩

lemma *synth-analz-fst*:
$\{\!|X,\ Y|\!\} \in synth\ (analz\ H) \Longrightarrow X \in synth\ (analz\ H)$
⟨*proof*⟩

lemma *synth-analz-snd*:
$\{\!|X,\ Y|\!\} \in synth\ (analz\ H) \Longrightarrow Y \in synth\ (analz\ H)$
⟨*proof*⟩

end

2 Protocol modeling and verification

theory *Protocol*
imports *Propaedeutics*
begin

2.1 Protocol modeling

The protocol under consideration can be formalized by means of the following inductive definition.

inductive-set *protocol* :: $(event\ list \times state \times msg\ set \times msg\ set)\ set$ **where**

Nil: $(\lfloor\rfloor,\ \mathit{start\text{-}S},\ \mathit{start\text{-}A},\ \mathit{start\text{-}U}) \in \mathit{protocol}\ |$

R1: $[\![(\mathit{evsR1},\ S,\ A,\ U) \in \mathit{protocol};\ \mathit{Pri\text{-}AgrK}\ s \notin U;\ s \neq 0;$
$\mathit{NonceS}\ (S\ (\mathit{Card}\ n,\ n,\ \mathit{run})) = \mathit{None}]\!]$
$\Longrightarrow (\mathit{Says}\ n\ \mathit{run}\ 1\ (\mathit{Card}\ n)\ (\mathit{User}\ m)\ (\mathit{Crypt}\ (\mathit{symK}\ n)\ (\mathit{Pri\text{-}AgrK}\ s))\ \#\ \mathit{evsR1},$
$S\ ((\mathit{Card}\ n,\ n,\ \mathit{run}) := S\ (\mathit{Card}\ n,\ n,\ \mathit{run})\ (\!|\mathit{NonceS} := \mathit{Some}\ s|\!)),$
$\mathit{if}\ n \in \mathit{bad}\ \mathit{then}\ \mathit{insert}\ (\mathit{Pri\text{-}AgrK}\ s)\ A\ \mathit{else}\ A,$
$\mathit{insert}\ (\mathit{Pri\text{-}AgrK}\ s)\ U) \in \mathit{protocol}\ |$

FR1: $[\![(\mathit{evsFR1},\ S,\ A,\ U) \in \mathit{protocol};\ \mathit{User}\ m \neq \mathit{Spy};\ s \neq 0;$
$\mathit{Crypt}\ (\mathit{symK}\ m)\ (\mathit{Pri\text{-}AgrK}\ s) \in \mathit{synth}\ (\mathit{analz}\ (A \cup \mathit{spies}\ \mathit{evsFR1}))]\!]$
$\Longrightarrow (\mathit{Says}\ n\ \mathit{run}\ 1\ \mathit{Spy}\ (\mathit{User}\ m)\ (\mathit{Crypt}\ (\mathit{symK}\ m)\ (\mathit{Pri\text{-}AgrK}\ s))\ \#\ \mathit{evsFR1},$
$S,\ A,\ U) \in \mathit{protocol}\ |$

C2: $[\![(\mathit{evsC2},\ S,\ A,\ U) \in \mathit{protocol};\ \mathit{Pri\text{-}AgrK}\ a \notin U;$
$\mathit{NonceS}\ (S\ (\mathit{User}\ m,\ n,\ \mathit{run})) = \mathit{None};$
$\mathit{Says}\ n\ \mathit{run}\ 1\ X\ (\mathit{User}\ m)\ (\mathit{Crypt}\ (\mathit{symK}\ n')\ (\mathit{Pri\text{-}AgrK}\ s)) \in \mathit{set}\ \mathit{evsC2};$
$s' = (\mathit{if}\ \mathit{symK}\ n' = \mathit{symK}\ m\ \mathit{then}\ s\ \mathit{else}\ 0)]\!]$
$\Longrightarrow (\mathit{Says}\ n\ \mathit{run}\ 2\ (\mathit{User}\ m)\ (\mathit{Card}\ n)\ (\mathit{pubAK}\ a)\ \#\ \mathit{evsC2},$
$S\ ((\mathit{User}\ m,\ n,\ \mathit{run}) := S\ (\mathit{User}\ m,\ n,\ \mathit{run})$
$(\!|\mathit{NonceS} := \mathit{Some}\ s',\ \mathit{IntMapK} := \mathit{Some}\ a|\!)),$
$\mathit{if}\ \mathit{User}\ m = \mathit{Spy}\ \mathit{then}\ \mathit{insert}\ (\mathit{Pri\text{-}AgrK}\ a)\ A\ \mathit{else}\ A,$
$\mathit{insert}\ (\mathit{Pri\text{-}AgrK}\ a)\ U) \in \mathit{protocol}\ |$

FC2: $[\![(\mathit{evsFC2},\ S,\ A,\ U) \in \mathit{protocol};$
$\mathit{Pri\text{-}AgrK}\ a \in \mathit{analz}\ (A \cup \mathit{spies}\ \mathit{evsFC2})]\!]$
$\Longrightarrow (\mathit{Says}\ n\ \mathit{run}\ 2\ \mathit{Spy}\ (\mathit{Card}\ n)\ (\mathit{pubAK}\ a)\ \#\ \mathit{evsFC2},$
$S,\ A,\ U) \in \mathit{protocol}\ |$

R2: $[\![(\mathit{evsR2},\ S,\ A,\ U) \in \mathit{protocol};\ \mathit{Pri\text{-}AgrK}\ b \notin U;$
$\mathit{NonceS}\ (S\ (\mathit{Card}\ n,\ n,\ \mathit{run})) \neq \mathit{None};$
$\mathit{IntMapK}\ (S\ (\mathit{Card}\ n,\ n,\ \mathit{run})) = \mathit{None};$
$\mathit{Says}\ n\ \mathit{run}\ 2\ X\ (\mathit{Card}\ n)\ (\mathit{pubAK}\ a) \in \mathit{set}\ \mathit{evsR2}]\!]$
$\Longrightarrow (\mathit{Says}\ n\ \mathit{run}\ 2\ (\mathit{Card}\ n)\ X\ (\mathit{pubAK}\ b)\ \#\ \mathit{evsR2},$
$S\ ((\mathit{Card}\ n,\ n,\ \mathit{run}) := S\ (\mathit{Card}\ n,\ n,\ \mathit{run})$
$(\!|\mathit{IntMapK} := \mathit{Some}\ b,\ \mathit{ExtMapK} := \mathit{Some}\ a|\!)),$
$A,\ \mathit{insert}\ (\mathit{Pri\text{-}AgrK}\ b)\ U) \in \mathit{protocol}\ |$

FR2: $[\![(\mathit{evsFR2},\ S,\ A,\ U) \in \mathit{protocol};\ \mathit{User}\ m \neq \mathit{Spy};$
$\mathit{Pri\text{-}AgrK}\ b \in \mathit{analz}\ (A \cup \mathit{spies}\ \mathit{evsFR2})]\!]$
$\Longrightarrow (\mathit{Says}\ n\ \mathit{run}\ 2\ \mathit{Spy}\ (\mathit{User}\ m)\ (\mathit{pubAK}\ b)\ \#\ \mathit{evsFR2},$
$S,\ A,\ U) \in \mathit{protocol}\ |$

C3: $[\![(\mathit{evsC3},\ S,\ A,\ U) \in \mathit{protocol};\ \mathit{Pri\text{-}AgrK}\ c \notin U;$
$\mathit{NonceS}\ (S\ (\mathit{User}\ m,\ n,\ \mathit{run})) = \mathit{Some}\ s;$
$\mathit{IntMapK}\ (S\ (\mathit{User}\ m,\ n,\ \mathit{run})) = \mathit{Some}\ a;$
$\mathit{ExtMapK}\ (S\ (\mathit{User}\ m,\ n,\ \mathit{run})) = \mathit{None};$
$\mathit{Says}\ n\ \mathit{run}\ 2\ X\ (\mathit{User}\ m)\ (\mathit{pubAK}\ b) \in \mathit{set}\ \mathit{evsC3};$
$c * (s + a * b) \neq 0]\!]$

$\Longrightarrow$ (*Says n run 3* (*User m*) (*Card n*) (*pubAK* (*c* $*$ (*s* $+$ *a* $*$ *b*))) $\#$ *evsC3*,
S ((*User m*, *n*, *run*) := *S* (*User m*, *n*, *run*)
(|*ExtMapK* := *Some b*, *IntAgrK* := *Some c*|)),
if User m $=$ *Spy then insert* (*Pri-AgrK c*) *A else A*,
insert (*Pri-AgrK c*) *U*) $\in$ *protocol* |

FC3: $[\![$(*evsFC3*, *S*, *A*, *U*) $\in$ *protocol*;
NonceS (*S* (*Card n*, *n*, *run*)) $=$ *Some s*;
IntMapK (*S* (*Card n*, *n*, *run*)) $=$ *Some b*;
ExtMapK (*S* (*Card n*, *n*, *run*)) $=$ *Some a*;
{*Pri-AgrK s*, *Pri-AgrK a*, *Pri-AgrK c*} $\subseteq$ *analz* (*A* $\cup$ *spies evsFC3*)$]\!]$
$\Longrightarrow$ (*Says n run 3 Spy* (*Card n*) (*pubAK* (*c* $*$ (*s* $+$ *a* $*$ *b*))) $\#$ *evsFC3*,
S, *A*, *U*) $\in$ *protocol* |

R3: $[\![$(*evsR3*, *S*, *A*, *U*) $\in$ *protocol*; *Pri-AgrK d* $\notin$ *U*;
NonceS (*S* (*Card n*, *n*, *run*)) $=$ *Some s*;
IntMapK (*S* (*Card n*, *n*, *run*)) $=$ *Some b*;
ExtMapK (*S* (*Card n*, *n*, *run*)) $=$ *Some a*;
IntAgrK (*S* (*Card n*, *n*, *run*)) $=$ *None*;
Says n run 3 X (*Card n*) (*pubAK* (*c* $*$ (*s'* $+$ *a* $*$ *b*))) $\in$ *set evsR3*;
Key (*sesK* (*c* $*$ *d* $*$ (*s'* $+$ *a* $*$ *b*))) $\notin$ *U*;
Key (*sesK* (*c* $*$ *d* $*$ (*s* $+$ *a* $*$ *b*))) $\notin$ *U*;
d $*$ (*s* $+$ *a* $*$ *b*) $\neq$ *0*$]\!]$
$\Longrightarrow$ (*Says n run 3* (*Card n*) *X* (*pubAK* (*d* $*$ (*s* $+$ *a* $*$ *b*))) $\#$ *evsR3*,
S ((*Card n*, *n*, *run*) := *S* (*Card n*, *n*, *run*)
(|*IntAgrK* := *Some d*, *ExtAgrK* := *Some* (*c* $*$ (*s'* $+$ *a* $*$ *b*))|)),
if s' $=$ *s* $\wedge$ *Pri-AgrK c* $\in$ *analz* (*A* $\cup$ *spies evsR3*)
then insert (*Key* (*sesK* (*c* $*$ *d* $*$ (*s* $+$ *a* $*$ *b*)))) *A else A*,
{*Pri-AgrK d*,
Key (*sesK* (*c* $*$ *d* $*$ (*s'* $+$ *a* $*$ *b*))), *Key* (*sesK* (*c* $*$ *d* $*$ (*s* $+$ *a* $*$ *b*))),
{|*Key* (*sesK* (*c* $*$ *d* $*$ (*s* $+$ *a* $*$ *b*))), *Agent X*, *Number n*, *Number run*|}} $\cup$
U) $\in$ *protocol* |

FR3: $[\![$(*evsFR3*, *S*, *A*, *U*) $\in$ *protocol*; *User m* $\neq$ *Spy*;
NonceS (*S* (*User m*, *n*, *run*)) $=$ *Some s*;
IntMapK (*S* (*User m*, *n*, *run*)) $=$ *Some a*;
ExtMapK (*S* (*User m*, *n*, *run*)) $=$ *Some b*;
IntAgrK (*S* (*User m*, *n*, *run*)) $=$ *Some c*;
{*Pri-AgrK s*, *Pri-AgrK b*, *Pri-AgrK d*} $\subseteq$ *analz* (*A* $\cup$ *spies evsFR3*);
Key (*sesK* (*c* $*$ *d* $*$ (*s* $+$ *a* $*$ *b*))) $\notin$ *U*$]\!]$
$\Longrightarrow$ (*Says n run 3 Spy* (*User m*) (*pubAK* (*d* $*$ (*s* $+$ *a* $*$ *b*))) $\#$ *evsFR3*, *S*,
insert (*Key* (*sesK* (*c* $*$ *d* $*$ (*s* $+$ *a* $*$ *b*)))) *A*,
{*Key* (*sesK* (*c* $*$ *d* $*$ (*s* $+$ *a* $*$ *b*))),
{|*Key* (*sesK* (*c* $*$ *d* $*$ (*s* $+$ *a* $*$ *b*))), *Agent* (*User m*), *Number n*, *Number run*|}} $\cup$ *U*) $\in$ *protocol* |

C4: $[\![$(*evsC4*, *S*, *A*, *U*) $\in$ *protocol*;
IntAgrK (*S* (*User m*, *n*, *run*)) $=$ *Some c*;
ExtAgrK (*S* (*User m*, *n*, *run*)) $=$ *None*;

$$
\begin{array}{l}
\quad Says\ n\ run\ 3\ X\ (User\ m)\ (pubAK\ f) \in set\ evsC4; \\
\quad \{\!|Key\ (sesK\ (c * f)),\ Agent\ (User\ m),\ Number\ n,\ Number\ run|\!\} \in U\rrbracket \\
\Longrightarrow (Says\ n\ run\ 4\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ (c * f))\ (pubAK\ f)) \#\ evsC4, \\
\quad S\ ((User\ m,\ n,\ run) := S\ (User\ m,\ n,\ run)\ (|ExtAgrK := Some\ f|)), \\
\quad A,\ U) \in protocol\ |
\end{array}
$$

$$
\begin{array}{ll}
FC4: & \llbracket(evsFC4,\ S,\ A,\ U) \in protocol; \\
& \quad NonceS\ (S\ (Card\ n,\ n,\ run)) = Some\ s; \\
& \quad IntMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ b; \\
& \quad ExtMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ a; \\
& \quad IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d; \\
& \quad ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e; \\
& \quad Crypt\ (sesK\ (d * e))\ (pubAK\ (d * (s + a * b))) \\
& \qquad \in synth\ (analz\ (A \cup spies\ evsFC4))\rrbracket \\
& \Longrightarrow (Says\ n\ run\ 4\ Spy\ (Card\ n) \\
& \quad (Crypt\ (sesK\ (d * e))\ (pubAK\ (d * (s + a * b)))) \#\ evsFC4, \\
& S,\ A,\ U) \in protocol\ |
\end{array}
$$

$$
\begin{array}{ll}
R4: & \llbracket(evsR4,\ S,\ A,\ U) \in protocol; \\
& \quad NonceS\ (S\ (Card\ n,\ n,\ run)) = Some\ s; \\
& \quad IntMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ b; \\
& \quad ExtMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ a; \\
& \quad IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d; \\
& \quad ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e; \\
& \quad Says\ n\ run\ 4\ X\ (Card\ n)\ (Crypt\ (sesK\ (d * e)) \\
& \qquad (pubAK\ (d * (s + a * b)))) \in set\ evsR4\rrbracket \\
& \Longrightarrow (Says\ n\ run\ 4\ (Card\ n)\ X\ (Crypt\ (sesK\ (d * e)) \\
& \quad \{\!|pubAK\ e,\ Auth\text{-}Data\ (priAK\ n)\ b,\ pubAK\ (priAK\ n), \\
& \quad\ Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ (priAK\ n)))|\!\}) \#\ evsR4, \\
& S,\ A,\ U) \in protocol\ |
\end{array}
$$

$$
\begin{array}{ll}
FR4: & \llbracket(evsFR4,\ S,\ A,\ U) \in protocol;\ User\ m \neq Spy; \\
& \quad NonceS\ (S\ (User\ m,\ n,\ run)) = Some\ s; \\
& \quad IntMapK\ (S\ (User\ m,\ n,\ run)) = Some\ a; \\
& \quad ExtMapK\ (S\ (User\ m,\ n,\ run)) = Some\ b; \\
& \quad IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c; \\
& \quad ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f; \\
& \quad Crypt\ (sesK\ (c * f)) \\
& \qquad \{\!|pubAK\ (c * (s + a * b)),\ Auth\text{-}Data\ g\ b,\ pubAK\ g, \\
& \quad Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ g))|\!\} \in synth\ (analz\ (A \cup spies\ evsFR4))\rrbracket \\
& \Longrightarrow (Says\ n\ run\ 4\ Spy\ (User\ m)\ (Crypt\ (sesK\ (c * f)) \\
& \quad \{\!|pubAK\ (c * (s + a * b)),\ Auth\text{-}Data\ g\ b,\ pubAK\ g, \\
& \quad\ Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ g))|\!\}) \#\ evsFR4, \\
& S,\ A,\ U) \in protocol\ |
\end{array}
$$

$$
\begin{array}{ll}
C5: & \llbracket(evsC5,\ S,\ A,\ U) \in protocol; \\
& \quad NonceS\ (S\ (User\ m,\ n,\ run)) = Some\ s; \\
& \quad IntMapK\ (S\ (User\ m,\ n,\ run)) = Some\ a; \\
& \quad ExtMapK\ (S\ (User\ m,\ n,\ run)) = Some\ b;
\end{array}
$$

$IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c;$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f;$
$Says\ n\ run\ 4\ X\ (User\ m)\ (Crypt\ (sesK\ (c * f))$
$\{|pubAK\ (c * (s + a * b)),\ Auth\text{-}Data\ g\ b,\ pubAK\ g,$
$Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ g))|\}) \in set\ evsC5[\![$
$\Longrightarrow (Says\ n\ run\ 5\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ \#\ evsC5,$
$S,\ A,\ U) \in protocol\ |$

$FC5:\ [\![(evsFC5,\ S,\ A,\ U) \in protocol;$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d;$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e;$
$Crypt\ (sesK\ (d * e))\ (Passwd\ n) \in synth\ (analz\ (A \cup spies\ evsFC5))]\!]$
$\Longrightarrow (Says\ n\ run\ 5\ Spy\ (Card\ n)\ (Crypt\ (sesK\ (d * e))\ (Passwd\ n))\ \#\ evsFC5,$
$S,\ A,\ U) \in protocol\ |$

$R5:\ [\![(evsR5,\ S,\ A,\ U) \in protocol;$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d;$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e;$
$Says\ n\ run\ 5\ X\ (Card\ n)\ (Crypt\ (sesK\ (d * e))\ (Passwd\ n)) \in set\ evsR5]\!]$
$\Longrightarrow (Says\ n\ run\ 5\ (Card\ n)\ X\ (Crypt\ (sesK\ (d * e))\ (Number\ 0))\ \#\ evsR5,$
$S,\ A,\ U) \in protocol\ |$

$FR5:\ [\![(evsFR5,\ S,\ A,\ U) \in protocol;\ User\ m \neq Spy;$
$IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c;$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f;$
$Crypt\ (sesK\ (c * f))\ (Number\ 0) \in synth\ (analz\ (A \cup spies\ evsFR5))]\!]$
$\Longrightarrow (Says\ n\ run\ 5\ Spy\ (User\ m)\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ \#\ evsFR5,$
$S,\ A,\ U) \in protocol$

Here below are some comments about the most significant points of this definition.

- Rules *R1* and *FR1* constrain the values of nonce s to be different from 0. In this way, the state of affairs where an incorrect PACE authentication key has been used to encrypt nonce s, so that a wrong value results from the decryption, can be modeled in rule *C2* by identifying such value with 0.

- The spy can disclose session keys as soon as they are established, namely in correspondence with rules *R3* and *FR3*.
  In the former rule, condition $s' = s$ identifies Diffie-Hellman private key c as the terminal's ephemeral private key for key agreement, and then $[c \times d \times (s + a \times b)]G$ as the terminal's value of the shared secret, which the spy can compute by multiplying the card's ephemeral public key $[d \times (s + a \times b)]G$ by c provided she knows c.
  In the latter rule, the spy is required to know private keys s, b, and d

to be able to compute and send public key $[d \times (s + a \times b)]G$. This is the only way to share with *User m* the same shared secret's value $[c \times d \times (s + a \times b)]G$, which the spy can compute by multiplying the user's ephemeral public key $[c \times (s + a \times b)]G$ by *d*.

- Rules *R3* and *FR3* record the user, the communication channel, and the protocol run associated to the session key having been established by adding this information to the set of the messages already used. In this way, rule *C4* can specify that the session key computed by *User m* is fresh by assuming that a corresponding record be included in set *U*. In fact, a simple check that the session key be not included in *U* would vacuously fail, as session keys are added to the set of the messages already used in rules *R3* and *FR3*.

2.2 Secrecy theorems

This section contains a series of lemmas culminating in the secrecy theorems *pr-key-secrecy* and *pr-passwd-secrecy*. Structured Isar proofs are used, possibly preceded by *apply*-style scripts in case a substantial amount of backward reasoning steps is required at the beginning.

lemma *pr-state*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$(NonceS\ (S\ (X,\ n,\ run)) = None \longrightarrow IntMapK\ (S\ (X,\ n,\ run)) = None) \wedge$
$(IntMapK\ (S\ (X,\ n,\ run)) = None \longrightarrow ExtMapK\ (S\ (X,\ n,\ run)) = None) \wedge$
$(ExtMapK\ (S\ (X,\ n,\ run)) = None \longrightarrow IntAgrK\ (S\ (X,\ n,\ run)) = None) \wedge$
$(IntAgrK\ (S\ (X,\ n,\ run)) = None \longrightarrow ExtAgrK\ (S\ (X,\ n,\ run)) = None)$
⟨*proof*⟩

lemma *pr-state-1*:
$[\![(evs,\ S,\ A,\ U) \in protocol;\ NonceS\ (S\ (X,\ n,\ run)) = None]\!] \Longrightarrow$
$IntMapK\ (S\ (X,\ n,\ run)) = None$
⟨*proof*⟩

lemma *pr-state-2*:
$[\![(evs,\ S,\ A,\ U) \in protocol;\ IntMapK\ (S\ (X,\ n,\ run)) = None]\!] \Longrightarrow$
$ExtMapK\ (S\ (X,\ n,\ run)) = None$
⟨*proof*⟩

lemma *pr-state-3*:
$[\![(evs,\ S,\ A,\ U) \in protocol;\ ExtMapK\ (S\ (X,\ n,\ run)) = None]\!] \Longrightarrow$
$IntAgrK\ (S\ (X,\ n,\ run)) = None$
⟨*proof*⟩

lemma *pr-state-4*:
$[\![(evs,\ S,\ A,\ U) \in protocol;\ IntAgrK\ (S\ (X,\ n,\ run)) = None]\!] \Longrightarrow$

$ExtAgrK\ (S\ (X,\ n,\ run)) = None$
⟨*proof*⟩

lemma *pr-analz-used*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow A \subseteq U$
⟨*proof*⟩

lemma *pr-key-parts-intro* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Key\ K \in parts\ (A \cup spies\ evs) \longrightarrow$
$Key\ K \in A$
⟨*proof*⟩

lemma *pr-key-analz*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow (Key\ K \in analz\ (A \cup spies\ evs)) = (Key\ K \in A)$
⟨*proof*⟩

lemma *pr-symk-used*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow Key\ (symK\ n) \in U$
⟨*proof*⟩

lemma *pr-symk-analz*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow (Key\ (symK\ n) \in analz\ (A \cup spies\ evs)) = (n \in bad)$
⟨*proof*⟩

lemma *pr-sesk-card* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d \longrightarrow$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e \longrightarrow$
$Key\ (sesK\ (d * e)) \in U$
⟨*proof*⟩

lemma *pr-sesk-user-1* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c \longrightarrow$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f \longrightarrow$
$\{\!|Key\ (sesK\ (c * f)),\ Agent\ (User\ m),\ Number\ n,\ Number\ run|\!\} \in U$
⟨*proof*⟩

lemma *pr-sesk-user-2* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\{\!|Key\ (sesK\ K),\ Agent\ (User\ m),\ Number\ n,\ Number\ run|\!\} \in U \longrightarrow$
$Key\ (sesK\ K) \in U$
⟨*proof*⟩

lemma *pr-auth-key-used*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow Pri\text{-}AgrK\ (priAK\ n) \in U$
⟨*proof*⟩

lemma *pr-int-mapk-used* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\quad IntMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ b \longrightarrow$
$\ Pri\text{-}AgrK\ b \in U$
⟨*proof*⟩

lemma *pr-valid-key-analz*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow Key\ (pubSK\ X) \in analz\ (A \cup spies\ evs)$
⟨*proof*⟩

lemma *pr-pri-agrk-parts* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\quad Pri\text{-}AgrK\ x \notin U \longrightarrow$
$\ Pri\text{-}AgrK\ x \notin parts\ (A \cup spies\ evs)$
⟨*proof*⟩

lemma *pr-pri-agrk-items*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\quad Pri\text{-}AgrK\ x \notin U \Longrightarrow$
$\ items\ (insert\ (Pri\text{-}AgrK\ x)\ (A \cup spies\ evs)) =$
$\quad insert\ (Pri\text{-}AgrK\ x)\ (items\ (A \cup spies\ evs))$
⟨*proof*⟩

lemma *pr-auth-data-items*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\ Pri\text{-}AgrK\ (priAK\ n) \notin items\ (A \cup spies\ evs) \wedge$
$\ (IntMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ b \longrightarrow$
$\quad Pri\text{-}AgrK\ b \notin items\ (A \cup spies\ evs))$
⟨*proof*⟩

lemma *pr-auth-key-analz*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow Pri\text{-}AgrK\ (priAK\ n) \notin analz\ (A \cup spies\ evs)$
⟨*proof*⟩

lemma *pr-int-mapk-analz*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\quad IntMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ b \Longrightarrow$
$\ Pri\text{-}AgrK\ b \notin analz\ (A \cup spies\ evs)$
⟨*proof*⟩

lemma *pr-key-set-unused* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\quad H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U \longrightarrow$
$\ analz\ (H \cup A \cup spies\ evs) = H \cup analz\ (A \cup spies\ evs)$
⟨*proof*⟩

lemma *pr-key-unused*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$

$Key\ K \notin U \Longrightarrow$
$analz\ (insert\ (Key\ K)\ (A \cup spies\ evs)) =$
$insert\ (Key\ K)\ (analz\ (A \cup spies\ evs))$
⟨*proof*⟩

lemma *pr-pri-agrk-unused*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Pri\text{-}AgrK\ x \notin U \Longrightarrow$
$analz\ (insert\ (Pri\text{-}AgrK\ x)\ (A \cup spies\ evs)) =$
$insert\ (Pri\text{-}AgrK\ x)\ (analz\ (A \cup spies\ evs))$
⟨*proof*⟩

lemma *pr-pri-agrk-analz-intro* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Pri\text{-}AgrK\ x \in analz\ (A \cup spies\ evs) \longrightarrow$
$Pri\text{-}AgrK\ x \in A$
⟨*proof*⟩

lemma *pr-pri-agrk-analz*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$(Pri\text{-}AgrK\ x \in analz\ (A \cup spies\ evs)) = (Pri\text{-}AgrK\ x \in A)$
⟨*proof*⟩

lemma *pr-ext-agrk-user-1* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$User\ m \neq Spy \longrightarrow$
$Says\ n\ run\ 4\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ K)\ (pubAK\ e)) \in set\ evs \longrightarrow$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) \neq None$
⟨*proof*⟩

lemma *pr-ext-agrk-user-2* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$User\ m \neq Spy \longrightarrow$
$Says\ n\ run\ 4\ X\ (User\ m)\ (Crypt\ (sesK\ K)$
$\{\!|pubAK\ e,\ Auth\text{-}Data\ x\ y,\ pubAK\ g,\ Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ g))|\!\})$
$\in set\ evs \longrightarrow$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) \neq None$
⟨*proof*⟩

lemma *pr-ext-agrk-user-3* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$User\ m \neq Spy \longrightarrow$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ e \longrightarrow$
$Says\ n\ run\ 4\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ K)\ (pubAK\ e')) \in set\ evs \longrightarrow$
$e' = e$
⟨*proof*⟩

lemma *pr-ext-agrk-user-4* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$

ExtAgrK (*S* (*User m*, *n*, *run*)) = *Some f* ⟶
(∃ *X*. *Says n run 3 X* (*User m*) (*pubAK f*) ∈ *set evs*)
⟨*proof*⟩

declare *fun-upd-apply* [*simp del*]

lemma *pr-ext-agrk-user-5* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
Says n run 3 X (*User m*) (*pubAK f*) ∈ *set evs* ⟶
(∃ *s a b d*. *f* = *d* ∗ (*s* + *a* ∗ *b*) ∧
NonceS (*S* (*Card n*, *n*, *run*)) = *Some s* ∧
IntMapK (*S* (*Card n*, *n*, *run*)) = *Some b* ∧
ExtMapK (*S* (*Card n*, *n*, *run*)) = *Some a* ∧
IntAgrK (*S* (*Card n*, *n*, *run*)) = *Some d* ∧
d ≠ *0* ∧ *s* + *a* ∗ *b* ≠ *0*) ∨
(∃ *b*. *Pri-AgrK b* ∈ *A* ∧
ExtMapK (*S* (*User m*, *n*, *run*)) = *Some b*)
(**is** - ⟹ *?H evs* ⟶ *?P S n run* ∨ *?Q S A n run*)
⟨*proof*⟩

declare *fun-upd-apply* [*simp*]

lemma *pr-int-agrk-user-1* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
IntAgrK (*S* (*User m*, *n*, *run*)) = *Some c* ⟶
Pri-AgrK c ∈ *U*
⟨*proof*⟩

lemma *pr-int-agrk-user-2* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
User m ≠ *Spy* ⟶
IntAgrK (*S* (*User m*, *n*, *run*)) = *Some c* ⟶
Pri-AgrK c ∉ *A*
⟨*proof*⟩

lemma *pr-int-agrk-user-3* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
NonceS (*S* (*User m*, *n*, *run*)) = *Some s* ⟶
IntMapK (*S* (*User m*, *n*, *run*)) = *Some a* ⟶
ExtMapK (*S* (*User m*, *n*, *run*)) = *Some b* ⟶
IntAgrK (*S* (*User m*, *n*, *run*)) = *Some c* ⟶
c ∗ (*s* + *a* ∗ *b*) ≠ *0*
⟨*proof*⟩

lemma *pr-int-agrk-card* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
NonceS (*S* (*Card n*, *n*, *run*)) = *Some s* ⟶
IntMapK (*S* (*Card n*, *n*, *run*)) = *Some b* ⟶
ExtMapK (*S* (*Card n*, *n*, *run*)) = *Some a* ⟶

IntAgrK (*S* (*Card n*, *n*, *run*)) = *Some d* ⟶
d ∗ (*s* + *a* ∗ *b*) ≠ *0*
⟨*proof*⟩

lemma *pr-ext-agrk-card* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
NonceS (*S* (*Card n*, *n*, *run*)) = *Some s* ⟶
IntMapK (*S* (*Card n*, *n*, *run*)) = *Some b* ⟶
ExtMapK (*S* (*Card n*, *n*, *run*)) = *Some a* ⟶
IntAgrK (*S* (*Card n*, *n*, *run*)) = *Some d* ⟶
ExtAgrK (*S* (*Card n*, *n*, *run*)) = *Some* (*c* ∗ (*s* + *a* ∗ *b*)) ⟶
Pri-AgrK c ∉ *A* ⟶
Key (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
⟨*proof*⟩

declare *fun-upd-apply* [*simp del*]

lemma *pr-sesk-user-3* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
{|*Key* (*sesK K*), *Agent* (*User m*), *Number n*, *Number run*|} ∈ *U* ⟶
Key (*sesK K*) ∈ *A* ⟶
(∃ *d e*. *K* = *d* ∗ *e* ∧
IntAgrK (*S* (*Card n*, *n*, *run*)) = *Some d* ∧
ExtAgrK (*S* (*Card n*, *n*, *run*)) = *Some e*) ∨
(∃ *b*. *Pri-AgrK b* ∈ *A* ∧
ExtMapK (*S* (*User m*, *n*, *run*)) = *Some b*)
(**is** - ⟹ *?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run*)
⟨*proof*⟩

declare *fun-upd-apply* [*simp*]

lemma *pr-sesk-auth* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
Crypt (*sesK K*) {|*pubAK e*, *Auth-Data x y*, *pubAK g*, *Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}
∈ *parts* (*A* ∪ *spies evs*) ⟶
Key (*sesK K*) ∈ *U*
⟨*proof*⟩

lemma *pr-sesk-passwd* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
Says n run 5 X (*Card n*) (*Crypt* (*sesK K*) (*Passwd m*)) ∈ *set evs* ⟶
Key (*sesK K*) ∈ *U*
⟨*proof*⟩

lemma *pr-sesk-card-user* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
User m ≠ *Spy* ⟶
NonceS (*S* (*User m*, *n*, *run*)) = *Some s* ⟶

$IntMapK\ (S\ (User\ m,\ n,\ run)) = Some\ a \longrightarrow$
$ExtMapK\ (S\ (User\ m,\ n,\ run)) = Some\ b \longrightarrow$
$IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c \longrightarrow$
$NonceS\ (S\ (Card\ n,\ n,\ run)) = Some\ s' \longrightarrow$
$IntMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ b' \longrightarrow$
$ExtMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ a' \longrightarrow$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d \longrightarrow$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ (c * (s + a * b)) \longrightarrow$
$s' + a' * b' = s + a * b \longrightarrow$
$Key\ (sesK\ (c * d * (s + a * b))) \notin A$
⟨*proof*⟩

lemma *pr-sign-key-used*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow Key\ (priSK\ X) \in U$
⟨*proof*⟩

lemma *pr-sign-key-analz*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow Key\ (priSK\ X) \notin analz\ (A \cup spies\ evs)$
⟨*proof*⟩

lemma *pr-auth-data-parts* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Auth\text{-}Data\ (priAK\ n)\ b \in parts\ (A \cup spies\ evs) \longrightarrow$
$(\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b)$
(**is** - $\Longrightarrow$ *?M* $\in$ - $\longrightarrow$ -)
⟨*proof*⟩

lemma *pr-sign-parts* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ g)) \in parts\ (A \cup spies\ evs) \longrightarrow$
$(\exists\, n.\ g = priAK\ n)$
(**is** - $\Longrightarrow$ *?M g* $\in$ - $\longrightarrow$ -)
⟨*proof*⟩

lemma *pr-key-secrecy-aux* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$User\ m \neq Spy \longrightarrow$
$NonceS\ (S\ (User\ m,\ n,\ run)) = Some\ s \longrightarrow$
$IntMapK\ (S\ (User\ m,\ n,\ run)) = Some\ a \longrightarrow$
$ExtMapK\ (S\ (User\ m,\ n,\ run)) = Some\ b \longrightarrow$
$IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c \longrightarrow$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f \longrightarrow$
$Says\ n\ run\ 4\ X\ (User\ m)\ (Crypt\ (sesK\ (c * f))$
$\{\!|pubAK\ (c * (s + a * b)),\ Auth\text{-}Data\ g\ b,\ pubAK\ g,$
$Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ g))|\!\}) \in set\ evs \longrightarrow$
$Key\ (sesK\ (c * f)) \notin A$
⟨*proof*⟩

theorem *pr-key-secrecy* [*rule-format*]:

$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$User\ m \neq Spy \longrightarrow$
$Says\ n\ run\ 5\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ K)\ (Passwd\ m)) \in set\ evs \longrightarrow$
$Key\ (sesK\ K) \notin analz\ (A \cup spies\ evs)$
⟨*proof*⟩

theorem *pr-passwd-secrecy* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$User\ m \neq Spy \longrightarrow$
$Passwd\ m \notin analz\ (A \cup spies\ evs)$
⟨*proof*⟩

2.3 Authenticity theorems

This section contains a series of lemmas culminating in the authenticity theorems *pr-user-authenticity* and *pr-card-authenticity*. Structured Isar proofs are used.

lemma *pr-passwd-parts* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Crypt\ (sesK\ K)\ (Passwd\ m) \in parts\ (A \cup spies\ evs) \longrightarrow$
$(\exists n\ run.\ Says\ n\ run\ 5\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ K)\ (Passwd\ m)) \in set\ evs) \vee$
$(\exists run.\ Says\ m\ run\ 5\ Spy\ (Card\ m)\ (Crypt\ (sesK\ K)\ (Passwd\ m)) \in set\ evs)$
(**is** - $\Longrightarrow$ *?M* $\in$ - $\longrightarrow$ *?P evs* $\vee$ *?Q evs*)
⟨*proof*⟩

lemma *pr-unique-run-1* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\{\!|Key\ (sesK\ (d * e)), Agent\ (User\ m), Number\ n', Number\ run'|\!\} \in U \longrightarrow$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d \longrightarrow$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e \longrightarrow$
$n' = n \wedge run' = run$
⟨*proof*⟩

lemma *pr-unique-run-2*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$IntAgrK\ (S\ (User\ m,\ n',\ run')) = Some\ c \Longrightarrow$
$ExtAgrK\ (S\ (User\ m,\ n',\ run')) = Some\ f \Longrightarrow$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d \Longrightarrow$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e \Longrightarrow$
$d * e = c * f \Longrightarrow$
$n' = n \wedge run' = run$
⟨*proof*⟩

lemma *pr-unique-run-3* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$IntAgrK\ (S\ (Card\ n',\ n',\ run')) = Some\ d' \longrightarrow$

$ExtAgrK\ (S\ (Card\ n',\ n',\ run')) = Some\ e' \longrightarrow$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d \longrightarrow$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e \longrightarrow$
$d * e = d' * e' \longrightarrow$
$n' = n \wedge run' = run$
⟨*proof*⟩

lemma *pr-unique-run-4* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Says\ n'\ run'\ 5\ X\ (Card\ n')\ (Crypt\ (sesK\ (d * e))\ (Passwd\ m)) \in set\ evs \longrightarrow$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d \longrightarrow$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e \longrightarrow$
$n' = n \wedge run' = run$
⟨*proof*⟩

theorem *pr-user-authenticity* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Says\ n\ run\ 5\ X\ (Card\ n)\ (Crypt\ (sesK\ K)\ (Passwd\ m)) \in set\ evs \longrightarrow$
$Says\ n\ run\ 5\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ K)\ (Passwd\ m)) \in set\ evs$
⟨*proof*⟩

lemma *pr-confirm-parts* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Crypt\ (sesK\ K)\ (Number\ 0) \in parts\ (A \cup spies\ evs) \longrightarrow$
$Key\ (sesK\ K) \notin A \longrightarrow$
$(\exists\, n\ run\ X.$
$Says\ n\ run\ 5\ X\ (Card\ n)\ (Crypt\ (sesK\ K)\ (Passwd\ n)) \in set\ evs \wedge$
$Says\ n\ run\ 5\ (Card\ n)\ X\ (Crypt\ (sesK\ K)\ (Number\ 0)) \in set\ evs)$
(**is** - ⟹ - ⟶ - ⟶ *?P K evs*)
⟨*proof*⟩

lemma *pr-confirm-says* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Says\ n\ run\ 5\ X\ Spy\ (Crypt\ (sesK\ K)\ (Number\ 0)) \in set\ evs \longrightarrow$
$Says\ n\ run\ 5\ Spy\ (Card\ n)\ (Crypt\ (sesK\ K)\ (Passwd\ n)) \in set\ evs$
⟨*proof*⟩

lemma *pr-passwd-says* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Says\ n\ run\ 5\ X\ (Card\ n)\ (Crypt\ (sesK\ K)\ (Passwd\ m)) \in set\ evs \longrightarrow$
$X = User\ m \vee X = Spy$
⟨*proof*⟩

lemma *pr-unique-run-5* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\{\!|Key\ (sesK\ K),\ Agent\ (User\ m'),\ Number\ n',\ Number\ run'|\!\} \in U \longrightarrow$
$\{\!|Key\ (sesK\ K),\ Agent\ (User\ m),\ Number\ n,\ Number\ run|\!\} \in U \longrightarrow$
$m = m' \wedge n = n' \wedge run = run'$
⟨*proof*⟩

lemma *pr-unique-run-6*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\{\!|Key\ (sesK\ (c * f)),\ Agent\ (User\ m'),\ Number\ n',\ Number\ run'|\!\} \in U \Longrightarrow$
$IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c \Longrightarrow$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f \Longrightarrow$
$m = m' \wedge n = n' \wedge run = run'$
⟨*proof*⟩

lemma *pr-unique-run-7* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Says\ n'\ run'\ 5\ (User\ m')\ (Card\ n')\ (Crypt\ (sesK\ K)\ (Passwd\ m')) \in set\ evs$
$\longrightarrow$
$\{\!|Key\ (sesK\ K),\ Agent\ (User\ m),\ Number\ n,\ Number\ run|\!\} \in U \longrightarrow$
$Key\ (sesK\ K) \notin A \longrightarrow$
$m' = m \wedge n' = n \wedge run' = run$
⟨*proof*⟩

lemma *pr-unique-run-8*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Says\ n'\ run'\ 5\ (User\ m')\ (Card\ n')\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m')) \in set\ evs \Longrightarrow$
$IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c \Longrightarrow$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f \Longrightarrow$
$Key\ (sesK\ (c * f)) \notin A \Longrightarrow$
$m' = m \wedge n' = n \wedge run' = run$
⟨*proof*⟩

lemma *pr-unique-passwd-parts* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Crypt\ (sesK\ K)\ (Passwd\ m') \in parts\ (A \cup spies\ evs) \longrightarrow$
$Crypt\ (sesK\ K)\ (Passwd\ m) \in parts\ (A \cup spies\ evs) \longrightarrow$
$m' = m$
⟨*proof*⟩

theorem *pr-card-authenticity* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Says\ n\ run\ 5\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ K)\ (Passwd\ m)) \in set\ evs \longrightarrow$
$Says\ n\ run\ 5\ X\ (User\ m)\ (Crypt\ (sesK\ K)\ (Number\ 0)) \in set\ evs \longrightarrow$
$n = m \wedge$
$(Says\ m\ run\ 5\ (Card\ m)\ (User\ m)\ (Crypt\ (sesK\ K)\ (Number\ 0)) \in set\ evs \vee$
$Says\ m\ run\ 5\ (Card\ m)\ Spy\ (Crypt\ (sesK\ K)\ (Number\ 0)) \in set\ evs)$
⟨*proof*⟩

end

References

[1] Bundesamt fur Sicherheit in der Informationstechnik. *Technical Guideline TR-03111 – Elliptic Curve Cryptography*, 2nd edition, 2012.

[2] International Civil Aviation Organization. *Doc 9303 – Machine Readable Travel Documents – Part 10: Logical Data Structure (LDS) for Storage of Biometrics and Other Data in the Contactless Integrated Circuit (IC)*, 7th edition, 2015.

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

[4] International Organization for Standardization. *ISO/IEC 7816-4 – Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*, 3rd edition, 2013.

[5] G. Kc and P. Karger. Preventing attacks on machine readable travel documents (mrtds). *IACR Cryptology ePrint Archive*, 2005.

[6] A. Krauss. *Defining Recursive Functions in Isabelle/HOL*. http://isabelle.in.tum.de/website-Isabelle2016/dist/Isabelle2016/doc/functions.pdf.

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

[8] T. Nipkow. *Programming and Proving in Isabelle/HOL*, Feb. 2016. http://isabelle.in.tum.de/website-Isabelle2016/dist/Isabelle2016/doc/prog-prove.pdf.

[9] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, Feb. 2016. http://isabelle.in.tum.de/website-Isabelle2016/dist/Isabelle2016/doc/tutorial.pdf.

[10] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, Dec. 1998.