# 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

March 17, 2025

**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 C$n$ for the $n$th command APDU, and step R$n$ for the $n$th 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*] ⇒ '*a* ∗ '*b* (‹(‹*indent=2 notation=‹mixfix message tuple›*›{|-,/ -|})›)
**syntax-consts**
  *-MTuple* ⇌ *MPair*
**translations**
  {|*x*, *y*, *z*|} ⇌ {|*x*, {|*y*, *z*|}|}
  {|*x*, *y*|} ⇌ *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*: $m \in bad \Longrightarrow symK\ n = symK\ m \Longrightarrow n \in bad$
**apply** (*rule-tac* $x = \{\}$ **in** *exI*)
**apply** (*rule-tac* $x = \lambda n.$ *if even n then n else Suc n* **in** *exI*)
**apply** (*rule-tac* $x = \lambda x.$ *int* (*inv nat-to-rat-surj x*) **in** *exI*)
**apply** (*rule-tac* $x = \lambda x.$ $2 *$ *inv nat-to-rat-surj x* **in** *exI*)
**apply** (*rule-tac* $x = \lambda n.$ *0* **in** *exI*)
**apply** (*rule-tac* $x = \lambda X.$ *Suc 0* **in** *exI*)
**proof** (*simp add*: *inj-on-def*, (*rule allI*)+, *rule impI*)
**fix** *x y*
**have** *surj nat-to-rat-surj*
**by** (*rule surj-nat-to-rat-surj*)

**hence** *inj* (*inv nat-to-rat-surj*)
**by** (*rule surj-imp-inj-inv*)
**moreover assume** *inv nat-to-rat-surj x* = *inv nat-to-rat-surj y*
**ultimately show** *x* = *y*
**by** (*rule injD*)
**qed**

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*]:
$Says\ i\ j\ k\ A\ B\ X \in set\ evs \longrightarrow X \in spies\ evs$
**apply** (*induction evs rule*: *spies.induct*)
**apply** *simp-all*
**done**

**lemma** *parts-subset*:
$H \subseteq parts\ H$
**by** (*rule subsetI*, *rule parts.Inj*)

**lemma** *parts-idem*:
$parts\ (parts\ H) = parts\ H$
**apply** (*rule equalityI*)
**apply** (*rule subsetI*)
**apply** (*erule parts.induct*)
**apply** *assumption*
**apply** (*erule parts.Fst*)
**apply** (*erule parts.Snd*)
**apply** (*erule parts.Body*)
**apply** (*erule parts.Auth-Fst*)
**apply** (*erule parts.Auth-Snd*)

**apply** (*rule parts-subset*)
**done**

**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-AgrK* $\cup$
    *range Pub-AgrK* $\Longrightarrow$
  *parts H* = *H*
**apply** (*rule equalityI* [*OF* - *parts-subset*])
**apply** (*rule subsetI*)
**apply** (*erule parts.induct*)
     **apply** *blast+*
**done**

**lemma** *parts-mono*:
 $G \subseteq H \Longrightarrow$ *parts* $G \subseteq$ *parts* $H$
**apply** (*rule subsetI*)
**apply** (*erule parts.induct*)
     **apply** (*drule subsetD*)
      **apply** *assumption*
     **apply** (*erule parts.Inj*)
    **apply** (*erule parts.Fst*)
   **apply** (*erule parts.Snd*)
  **apply** (*erule parts.Body*)
 **apply** (*erule parts.Auth-Fst*)
**apply** (*erule parts.Auth-Snd*)
**done**

**lemma** *parts-insert*:
 *insert X* (*parts H*) $\subseteq$ *parts* (*insert X H*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
 **apply** *simp*
 **apply** (*rule parts.Inj*)
 **apply** *simp*
**apply** (*erule rev-subsetD*)
**apply** (*rule parts-mono*)
**apply** *blast*
**done**

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

```
    range Nonce ∪
    range Key ∪
    range Hash ∪
    range Passwd ∪
    range Pri-AgrK ∪
    range Pub-AgrK ⟹
  parts (insert X H) = insert X (parts H)
apply (rule equalityI [OF - parts-insert])
apply (rule subsetI)
apply (erule parts.induct)
     apply simp-all
     apply (rotate-tac [!])
     apply (erule disjE)
      apply simp
     apply (rule disjI2)
     apply (erule parts.Inj)
    apply (erule disjE)
     apply blast
    apply (rule disjI2)
    apply (erule parts.Fst)
   apply (erule disjE)
    apply blast
   apply (rule disjI2)
   apply (erule parts.Snd)
  apply (erule disjE)
   apply blast
  apply (rule disjI2)
  apply (erule parts.Body)
 apply (erule disjE)
  apply blast
 apply (rule disjI2)
 apply (erule parts.Auth-Fst)
apply (erule disjE)
 apply blast
apply (rule disjI2)
apply (erule parts.Auth-Snd)
done

lemma parts-auth-data-1:
 parts (insert (Auth-Data x y) H) ⊆
    {Pri-AgrK x, Pri-AgrK y, Auth-Data x y} ∪ parts H
apply (rule subsetI)
apply (erule parts.induct)
     apply simp-all
     apply (erule disjE)
      apply simp
     apply (rule-tac [1-4] disjI2)+
     apply (erule parts.Inj)
    apply (erule parts.Fst)
```

**apply** (*erule parts.Snd*)
**apply** (*erule parts.Body*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule disjI2*)+
**apply** (*erule parts.Auth-Fst*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule disjI2*)+
**apply** (*erule parts.Auth-Snd*)
**done**

**lemma** *parts-auth-data-2*:
{*Pri-AgrK x*, *Pri-AgrK y*, *Auth-Data x y*} ∪ *parts H* ⊆
*parts* (*insert* (*Auth-Data x y*) *H*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Auth-Fst* [*of* - *y*])
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Auth-Snd* [*of x*])
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*erule rev-subsetD*)
**apply** (*rule parts-mono*)
**apply** *blast*
**done**

**lemma** *parts-auth-data*:
*parts* (*insert* (*Auth-Data x y*) *H*) =
{*Pri-AgrK x*, *Pri-AgrK y*, *Auth-Data x y*} ∪ *parts H*
**by** (*rule equalityI*, *rule parts-auth-data-1*, *rule parts-auth-data-2*)

**lemma** *parts-crypt-1*:
*parts* (*insert* (*Crypt K X*) *H*) ⊆ *insert* (*Crypt K X*) (*parts* (*insert X H*))
**apply** (*rule subsetI*)
**apply** (*erule parts.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1*−*3*] *disjI2*)

**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*erule parts.Fst*)
**apply** (*erule parts.Snd*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*rule disjI2*)
**apply** (*erule parts.Body*)
**apply** (*erule parts.Auth-Fst*)
**apply** (*erule parts.Auth-Snd*)
**done**

**lemma** *parts-crypt-2*:
*insert* (*Crypt K X*) (*parts* (*insert X H*)) ⊆ *parts* (*insert* (*Crypt K X*) *H*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*subst parts-idem* [*symmetric*])
**apply** (*erule rev-subsetD*)
**apply** (*rule parts-mono*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Body* [*of K*])
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*rule parts.Inj*)
**apply** *simp*
**done**

**lemma** *parts-crypt*:
*parts* (*insert* (*Crypt K X*) *H*) = *insert* (*Crypt K X*) (*parts* (*insert X H*))
**by** (*rule equalityI*, *rule parts-crypt-1*, *rule parts-crypt-2*)

**lemma** *parts-mpair-1*:
*parts* (*insert* {|*X*, *Y*|} *H*) ⊆ *insert* {|*X*, *Y*|} (*parts* ({*X*, *Y*} ∪ *H*))
**apply** (*rule subsetI*)
**apply** (*erule parts.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*rule parts.Inj*)

**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*erule parts.Fst*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*erule parts.Snd*)
**apply** (*erule parts.Body*)
**apply** (*erule parts.Auth-Fst*)
**apply** (*erule parts.Auth-Snd*)
**done**

**lemma** *parts-mpair-2*:
*insert* {|*X*, *Y*|} (*parts* ({*X*, *Y*} ∪ *H*)) ⊆ *parts* (*insert* {|*X*, *Y*|} *H*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*subst parts-idem* [*symmetric*])
**apply** (*erule rev-subsetD*)
**apply** (*rule parts-mono*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Fst* [*of - Y*])
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule parts.Snd* [*of X*])
**apply** (*rule parts.Inj*)
**apply** *simp*
**apply** (*rule parts.Inj*)
**apply** *simp*
**done**

**lemma** *parts-mpair*:
*parts* (*insert* {|*X*, *Y*|} *H*) = *insert* {|*X*, *Y*|} (*parts* ({*X*, *Y*} ∪ *H*))
**by** (*rule equalityI*, *rule parts-mpair-1*, *rule parts-mpair-2*)

**lemma** *items-subset*:
*H* ⊆ *items H*
**by** (*rule subsetI*, *rule items.Inj*)

**lemma** *items-idem*:
*items* (*items H*) = *items H*
**apply** (*rule equalityI*)
**apply** (*rule subsetI*)
**apply** (*erule items.induct*)
**apply** *assumption*
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**apply** (*rule items-subset*)
**done**

**lemma** *items-parts-subset*:
*items H* ⊆ *parts H*
**apply** (*rule subsetI*)
**apply** (*erule items.induct*)
**apply** (*erule parts.Inj*)
**apply** (*erule parts.Fst*)
**apply** (*erule parts.Snd*)
**apply** (*erule parts.Body*)
**apply** (*erule parts.Auth-Fst*)
**apply** (*erule parts.Auth-Snd*)
**done**

**lemma** *items-simp*:
*H* ⊆ *range Agent* ∪
*range Number* ∪
*range Nonce* ∪
*range Key* ∪
*range Hash* ∪
*range Passwd* ∪
*range Pri-AgrK* ∪
*range Pub-AgrK* ⟹
*items H* = *H*
**apply** (*rule equalityI*)
**apply** (*subst* (*3*) *parts-simp* [*symmetric*])
**apply** *assumption*
**apply** (*rule items-parts-subset*)
**apply** (*rule items-subset*)
**done**

**lemma** *items-mono*:
*G* ⊆ *H* ⟹ *items G* ⊆ *items H*
**apply** (*rule subsetI*)

**apply** (*erule items.induct*)
**apply** (*drule subsetD*)
**apply** *assumption*
**apply** (*erule items.Inj*)
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *items-insert*:
*insert X* (*items H*) ⊆ *items* (*insert X H*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule rev-subsetD*)
**apply** (*rule items-mono*)
**apply** *blast*
**done**

**lemma** *items-simp-insert-1*:
*X* ∈ *items H* ⟹ *items* (*insert X H*) = *items H*
**apply** (*rule equalityI*)
**apply** (*rule subsetI*)
**apply** (*erule items.induct* [*of* - *insert X H*])
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*erule items.Inj*)
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**apply** (*rule items-mono*)
**apply** *blast*
**done**

**lemma** *items-simp-insert-2*:
*X* ∈ *range Agent* ∪
*range Number* ∪

```
    range Nonce ∪
    range Key ∪
    range Hash ∪
    range Passwd ∪
    range Pub-AgrK ⟹
  items (insert X H) = insert X (items H)
apply (rule equalityI [OF - items-insert])
apply (rule subsetI)
apply (erule items.induct)
     apply simp-all
     apply (rotate-tac [!])
     apply (erule disjE)
      apply simp
     apply (rule disjI2)
     apply (erule items.Inj)
    apply (erule disjE)
     apply blast
    apply (rule disjI2)
    apply (erule items.Fst)
   apply (erule disjE)
    apply blast
   apply (rule disjI2)
   apply (erule items.Snd)
  apply (erule disjE)
   apply blast
  apply (rule disjI2)
  apply (erule items.Body)
 apply (erule disjE)
  apply blast
 apply (erule disjE)
  apply blast
 apply (rule disjI2)
 apply (erule items.Auth-Fst)
 apply assumption
apply (erule disjE)
 apply blast
apply (erule disjE)
 apply blast
apply (rule disjI2)
apply (erule items.Auth-Snd)
apply assumption
done

lemma items-pri-agrk-out:
 Pri-AgrK x ∉ parts H ⟹
    items (insert (Pri-AgrK x) H) = insert (Pri-AgrK x) (items H)
apply (rule equalityI [OF - items-insert])
apply (rule subsetI)
apply (erule items.induct)
```

**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*erule items.Inj*)
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*drule subsetD* [*OF items-parts-subset* [*of H*]])
**apply** (*drule parts.Auth-Snd*)
**apply** *simp*
**apply** (*rule disjI2*)
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*drule subsetD* [*OF items-parts-subset* [*of H*]])
**apply** (*drule parts.Auth-Fst*)
**apply** *simp*
**apply** (*rule disjI2*)
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *items-auth-data-in-1*:
*items* (*insert* (*Auth-Data x y*) *H*) ⊆
*insert* (*Auth-Data x y*) (*items* ({*Pri-AgrK x*, *Pri-AgrK y*} ∪ *H*))
**apply** (*rule subsetI*)
**apply** (*erule items.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Inj*)

**apply** *simp*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *items-auth-data-in-2*:
*Pri-AgrK x* ∈ *items H* ∨ *Pri-AgrK y* ∈ *items H* ⟹
*insert* (*Auth-Data x y*) (*items* ({*Pri-AgrK x*, *Pri-AgrK y*} ∪ *H*)) ⊆
*items* (*insert* (*Auth-Data x y*) *H*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** *rotate-tac*
**apply** (*erule disjE*)
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*subst items-idem* [*symmetric*])
**apply** (*erule rev-subsetD*)
**apply** (*rule items-mono*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** *rotate-tac*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** (*erule rev-subsetD*)
**apply** (*rule items-mono*)
**apply** *blast*
**apply** (*rule items.Auth-Fst* [*of* - *y*])
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule rev-subsetD*)
**apply** (*rule items-mono*)
**apply** *blast*
**apply** *rotate-tac*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** (*rule items.Auth-Snd* [*of x*])
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule rev-subsetD*)
**apply** (*rule items-mono*)
**apply** *blast*
**apply** (*erule rev-subsetD*)
**apply** (*rule items-mono*)
**apply** *blast*
**apply** (*rule items.Inj*)
**apply** *simp*
**done**

**lemma** *items-auth-data-in*:
*Pri-AgrK x* ∈ *items H* ∨ *Pri-AgrK y* ∈ *items H* ⟹
*items* (*insert* (*Auth-Data x y*) *H*) =
*insert* (*Auth-Data x y*) (*items* ({*Pri-AgrK x*, *Pri-AgrK y*} ∪ *H*))
**by** (*rule equalityI*, *rule items-auth-data-in-1*, *rule items-auth-data-in-2*)

**lemma** *items-auth-data-out*:
⟦*Pri-AgrK x* ∉ *items H*; *Pri-AgrK y* ∉ *items H*⟧ ⟹
*items* (*insert* (*Auth-Data x y*) *H*) = *insert* (*Auth-Data x y*) (*items H*)
**apply** (*rule equalityI* [*OF - items-insert*])
**apply** (*rule subsetI*)
**apply** (*erule items.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*erule items.Inj*)
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *items-crypt-1*:
*items* (*insert* (*Crypt K X*) *H*) ⊆ *insert* (*Crypt K X*) (*items* (*insert X H*))
**apply** (*rule subsetI*)
**apply** (*erule items.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.Body*)
**apply** (*erule items.Auth-Fst*)

**apply** *assumption*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *items-crypt-2*:
*insert* (*Crypt K X*) (*items* (*insert X H*)) ⊆ *items* (*insert* (*Crypt K X*) *H*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.induct*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Body* [*of K*])
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *items-crypt*:
*items* (*insert* (*Crypt K X*) *H*) = *insert* (*Crypt K X*) (*items* (*insert X H*))
**by** (*rule equalityI*, *rule items-crypt-1*, *rule items-crypt-2*)

**lemma** *items-mpair-1*:
*items* (*insert* {|*X*, *Y*|} *H*) ⊆ *insert* {|*X*, *Y*|} (*items* ({*X*, *Y*} ∪ *H*))
**apply** (*rule subsetI*)
**apply** (*erule items.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Inj*)
**apply** *simp*

**apply** (*erule items.Fst*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *items-mpair-2*:
*insert* $\{\!|X,\ Y|\!\}$ (*items* ($\{X,\ Y\} \cup H$)) $\subseteq$ *items* (*insert* $\{\!|X,\ Y|\!\}$ $H$)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.induct*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Fst* [*of* - $Y$])
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule items.Snd* [*of* $X$])
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*rule items.Inj*)
**apply** *simp*
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *items-mpair*:
*items* (*insert* $\{\!|X,\ Y|\!\}$ $H$) = *insert* $\{\!|X,\ Y|\!\}$ (*items* ($\{X,\ Y\} \cup H$))
**by** (*rule equalityI*, *rule items-mpair-1*, *rule items-mpair-2*)

**lemma** *analz-subset*:
$H \subseteq$ *analz* $H$

**by** (*rule subsetI*, *rule analz.Inj*)

**lemma** *analz-idem*:
*analz* (*analz H*) = *analz H*
**apply** (*rule equalityI*)
**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
**apply** *assumption*
**apply** (*erule analz.Fst*)
**apply** (*erule analz.Snd*)
**apply** (*erule analz.Decrypt*)
**apply** *assumption*
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**apply** (*rule analz-subset*)
**done**

**lemma** *analz-parts-subset*:
*analz H* ⊆ *parts H*
**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
**apply** (*erule parts.Inj*)
**apply** (*erule parts.Fst*)
**apply** (*erule parts.Snd*)
**apply** (*erule parts.Body*)
**apply** (*erule parts.Auth-Fst*)
**apply** (*erule parts.Auth-Snd*)
**done**

**lemma** *analz-items-subset*:
*analz H* ⊆ *items H*
**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
**apply** (*erule items.Inj*)
**apply** (*erule items.Fst*)
**apply** (*erule items.Snd*)
**apply** (*erule items.Body*)
**apply** (*erule items.Auth-Fst*)
**apply** *assumption*
**apply** (*erule items.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-simp*:
*H* ⊆ *range Agent* ∪
*range Number* ∪
*range Nonce* ∪

```
    range Key ∪
    range Hash ∪
    range Passwd ∪
    range Pri-AgrK ∪
    range Pub-AgrK ⟹
  analz H = H
apply (rule equalityI)
 apply (subst (3) parts-simp [symmetric])
  apply assumption
 apply (rule analz-parts-subset)
apply (rule analz-subset)
done

lemma analz-mono:
 G ⊆ H ⟹ analz G ⊆ analz H
apply (rule subsetI)
apply (erule analz.induct)
     apply (drule subsetD)
      apply assumption
     apply (erule analz.Inj)
    apply (erule analz.Fst)
   apply (erule analz.Snd)
  apply (erule analz.Decrypt)
  apply assumption
 apply (erule analz.Auth-Fst)
 apply assumption
apply (erule analz.Auth-Snd)
apply assumption
done

lemma analz-insert:
 insert X (analz H) ⊆ analz (insert X H)
apply (rule subsetI)
apply simp
apply (erule disjE)
 apply simp
 apply (rule analz.Inj)
 apply simp
apply (erule rev-subsetD)
apply (rule analz-mono)
apply blast
done

lemma analz-simp-insert-1:
 X ∈ analz H ⟹ analz (insert X H) = analz H
apply (rule equalityI)
 apply (rule subsetI)
 apply (erule analz.induct [of - insert X H])
     apply simp
```

**apply** (*erule disjE*)
**apply** *simp*
**apply** (*erule analz.Inj*)
**apply** (*erule analz.Fst*)
**apply** (*erule analz.Snd*)
**apply** (*erule analz.Decrypt*)
**apply** *assumption*
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**apply** (*rule analz-mono*)
**apply** *blast*
**done**

**lemma** *analz-simp-insert-2*:
*X* ∈ *range Agent* ∪
*range Number* ∪
*range Nonce* ∪
*range Hash* ∪
*range Passwd* ∪
*range Pub-AgrK* ⟹
*analz* (*insert X H*) = *insert X* (*analz H*)
**apply** (*rule equalityI* [*OF - analz-insert*])
**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
**apply** *simp-all*
**apply** (*rotate-tac* [!])
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule disjI2*)
**apply** (*erule analz.Inj*)
**apply** (*erule disjE*)
**apply** *blast*
**apply** (*rule disjI2*)
**apply** (*erule analz.Fst*)
**apply** (*erule disjE*)
**apply** *blast*
**apply** (*rule disjI2*)
**apply** (*erule analz.Snd*)
**apply** (*erule disjE*)
**apply** *blast*
**apply** (*erule disjE*)
**apply** *blast*
**apply** (*rule disjI2*)
**apply** (*erule analz.Decrypt*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *blast*

**apply** (*erule disjE*)
**apply** *blast*
**apply** (*rule disjI2*)
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *blast*
**apply** (*erule disjE*)
**apply** *blast*
**apply** (*rule disjI2*)
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-auth-data-in-1*:
*analz* (*insert* (*Auth-Data x y*) *H*) ⊆
*insert* (*Auth-Data x y*) (*analz* ({*Pri-AgrK x*, *Pri-AgrK y*} ∪ *H*))
**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.Fst*)
**apply** (*erule analz.Snd*)
**apply** (*erule analz.Decrypt*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-auth-data-in-2*:
*Pri-AgrK x* ∈ *analz H* ∨ *Pri-AgrK y* ∈ *analz H* ⟹
*insert* (*Auth-Data x y*) (*analz* ({*Pri-AgrK x*, *Pri-AgrK y*} ∪ *H*)) ⊆
*analz* (*insert* (*Auth-Data x y*) *H*)
**apply** (*rule subsetI*)
**apply** *simp*

**apply** *rotate-tac*
**apply** (*erule disjE*)
 **apply** (*rule analz.Inj*)
 **apply** *simp*
**apply** (*subst analz-idem* [*symmetric*])
**apply** (*erule rev-subsetD*)
**apply** (*rule analz-mono*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** *rotate-tac*
**apply** (*erule disjE*)
 **apply** *simp*
 **apply** (*erule disjE*)
  **apply** (*erule rev-subsetD*)
  **apply** (*rule analz-mono*)
  **apply** *blast*
 **apply** (*rule analz.Auth-Fst* [*of* - *y*])
  **apply** (*rule analz.Inj*)
  **apply** *simp*
 **apply** (*erule rev-subsetD*)
 **apply** (*rule analz-mono*)
 **apply** *blast*
**apply** *rotate-tac*
**apply** (*erule disjE*)
 **apply** *simp*
 **apply** (*erule disjE*)
  **apply** (*rule analz.Auth-Snd* [*of x*])
   **apply** (*rule analz.Inj*)
   **apply** *simp*
  **apply** (*erule rev-subsetD*)
  **apply** (*rule analz-mono*)
  **apply** *blast*
 **apply** (*erule rev-subsetD*)
 **apply** (*rule analz-mono*)
 **apply** *blast*
**apply** (*rule analz.Inj*)
**apply** *simp*
**done**

**lemma** *analz-auth-data-in*:
 *Pri-AgrK x* ∈ *analz H* ∨ *Pri-AgrK y* ∈ *analz H* ⟹
    *analz* (*insert* (*Auth-Data x y*) *H*) =
      *insert* (*Auth-Data x y*) (*analz* ({*Pri-AgrK x*, *Pri-AgrK y*} ∪ *H*))
**by** (*rule equalityI*, *rule analz-auth-data-in-1*, *rule analz-auth-data-in-2*)

**lemma** *analz-auth-data-out*:
 ⟦*Pri-AgrK x* ∉ *analz H*; *Pri-AgrK y* ∉ *analz H*⟧ ⟹
    *analz* (*insert* (*Auth-Data x y*) *H*) = *insert* (*Auth-Data x y*) (*analz H*)
**apply** (*rule equalityI* [*OF* - *analz-insert*])

**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*erule analz.Inj*)
**apply** (*erule analz.Fst*)
**apply** (*erule analz.Snd*)
**apply** (*erule analz.Decrypt*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-crypt-in-1*:
*analz* (*insert* (*Crypt K X*) *H*) ⊆ *insert* (*Crypt K X*) (*analz* (*insert X H*))
**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1−4*] *disjI2*)
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.Fst*)
**apply** (*erule analz.Snd*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.Decrypt*)
**apply** *assumption*
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-crypt-in-2*:
*Key* (*invK K*) ∈ *analz H* ⟹
*insert* (*Crypt K X*) (*analz* (*insert X H*)) ⊆ *analz* (*insert* (*Crypt K X*) *H*)
**apply** (*rule subsetI*)

**apply** *simp*
**apply** (*erule disjE*)
 **apply** *simp*
 **apply** (*rule analz.Inj*)
 **apply** *simp*
**apply** *rotate-tac*
**apply** (*erule analz.induct*)
     **apply** *simp*
     **apply** (*erule disjE*)
      **apply** *simp*
      **apply** (*rule analz.Decrypt* [*of K*])
       **apply** (*rule analz.Inj*)
       **apply** *simp*
      **apply** (*erule rev-subsetD*)
      **apply** (*rule analz-mono*)
      **apply** *blast*
     **apply** (*rule analz.Inj*)
     **apply** *simp*
    **apply** (*erule analz.Fst*)
   **apply** (*erule analz.Snd*)
  **apply** (*erule analz.Decrypt*)
  **apply** *assumption*
 **apply** (*erule analz.Auth-Fst*)
 **apply** *assumption*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-crypt-in*:
 *Key* (*invK K*) ∈ *analz H* ⟹
    *analz* (*insert* (*Crypt K X*) *H*) = *insert* (*Crypt K X*) (*analz* (*insert X H*))
**by** (*rule equalityI*, *rule analz-crypt-in-1*, *rule analz-crypt-in-2*)

**lemma** *analz-crypt-out*:
 *Key* (*invK K*) ∉ *analz H* ⟹
    *analz* (*insert* (*Crypt K X*) *H*) = *insert* (*Crypt K X*) (*analz H*)
**apply** (*rule equalityI* [*OF - analz-insert*])
**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
     **apply** *simp-all*
     **apply** (*erule disjE*)
      **apply** *simp*
     **apply** (*rule-tac* [*1−4*] *disjI2*)
     **apply** (*erule analz.Inj*)
    **apply** (*erule analz.Fst*)
   **apply** (*erule analz.Snd*)
  **apply** (*erule disjE*)
   **apply** *simp*
  **apply** (*erule analz.Decrypt*)

**apply** *assumption*
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-mpair-1*:
*analz* (*insert* ⦃*X*, *Y*⦄ *H*) ⊆ *insert* ⦃*X*, *Y*⦄ (*analz* ({*X*, *Y*} ∪ *H*))
**apply** (*rule subsetI*)
**apply** (*erule analz.induct*)
**apply** *simp-all*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule-tac* [*1*−*4*] *disjI2*)
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.Fst*)
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.Snd*)
**apply** (*erule analz.Decrypt*)
**apply** *assumption*
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-mpair-2*:
*insert* ⦃*X*, *Y*⦄ (*analz* ({*X*, *Y*} ∪ *H*)) ⊆ *analz* (*insert* ⦃*X*, *Y*⦄ *H*)
**apply** (*rule subsetI*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.induct*)
**apply** *simp*
**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule analz.Fst* [*of* - *Y*])
**apply** (*rule analz.Inj*)
**apply** *simp*

**apply** (*erule disjE*)
**apply** *simp*
**apply** (*rule analz.Snd* [*of X*])
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*rule analz.Inj*)
**apply** *simp*
**apply** (*erule analz.Fst*)
**apply** (*erule analz.Snd*)
**apply** (*erule analz.Decrypt*)
**apply** *assumption*
**apply** (*erule analz.Auth-Fst*)
**apply** *assumption*
**apply** (*erule analz.Auth-Snd*)
**apply** *assumption*
**done**

**lemma** *analz-mpair*:
*analz* (*insert* {|*X*, *Y*|} *H*) = *insert* {|*X*, *Y*|} (*analz* ({*X*, *Y*} ∪ *H*))
**by** (*rule equalityI*, *rule analz-mpair-1*, *rule analz-mpair-2*)

**lemma** *synth-simp-intro*:
*X* ∈ *synth H* ⟹
*X* ∈ *range Nonce* ∪
*range Key* ∪
*range Passwd* ∪
*range Pri-AgrK* ∪
*range Pub-AgrK* ⟹
*X* ∈ *H*
**by** (*erule synth.cases*, *blast*+)

**lemma** *synth-auth-data*:
*Auth-Data x y* ∈ *synth H* ⟹
*Auth-Data x y* ∈ *H* ∨ *Pri-AgrK x* ∈ *H* ∧ *Pri-AgrK y* ∈ *H*
**by** (*erule synth.cases*, *simp-all*)

**lemma** *synth-crypt*:
*Crypt K X* ∈ *synth H* ⟹ *Crypt K X* ∈ *H* ∨ *X* ∈ *synth H* ∧ *Key K* ∈ *H*
**by** (*erule synth.cases*, *simp-all*)

**lemma** *synth-mpair*:
{|*X*, *Y*|} ∈ *synth H* ⟹ {|*X*, *Y*|} ∈ *H* ∨ *X* ∈ *synth H* ∧ *Y* ∈ *synth H*
**by** (*erule synth.cases*, *simp-all*)

**lemma** *synth-analz-fst*:
{|*X*, *Y*|} ∈ *synth* (*analz H*) ⟹ *X* ∈ *synth* (*analz H*)
**proof** (*drule-tac synth-mpair*, *erule-tac disjE*)
**qed** (*drule analz.Fst*, *erule synth.Inj*, *erule conjE*)

**lemma** *synth-analz-snd*:
$\{\!|X,\ Y|\!\} \in synth\ (analz\ H) \Longrightarrow Y \in synth\ (analz\ H)$
**proof** (*drule-tac synth-mpair*, *erule-tac disjE*)
**qed** (*drule analz.Snd*, *erule synth.Inj*, *erule conjE*)

**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* × *state* × *msg set* × *msg set*) *set* **where**

*Nil*: $([],\ start\text{-}S,\ start\text{-}A,\ start\text{-}U) \in protocol\ |$

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

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

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

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

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

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

*C3*: $[\![(evsC3,\ S,\ A,\ U) \in protocol;\ Pri\text{-}AgrK\ c \notin U;$
$NonceS\ (S\ (User\ m,\ n,\ run)) = Some\ s;$
$IntMapK\ (S\ (User\ m,\ n,\ run)) = Some\ a;$
$ExtMapK\ (S\ (User\ m,\ n,\ run)) = None;$
$Says\ n\ run\ 2\ X\ (User\ m)\ (pubAK\ b) \in set\ evsC3;$
$c * (s + a * b) \neq 0]\!]$
$\Longrightarrow (Says\ n\ run\ 3\ (User\ m)\ (Card\ n)\ (pubAK\ (c * (s + a * b)))\ \#\ evsC3,$
$S\ ((User\ m,\ n,\ run) := S\ (User\ m,\ n,\ run)$
$(\!|ExtMapK := Some\ b,\ IntAgrK := Some\ c|\!)),$
$if\ User\ m = Spy\ then\ insert\ (Pri\text{-}AgrK\ c)\ A\ else\ A,$
$insert\ (Pri\text{-}AgrK\ c)\ U) \in protocol\ |$

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

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

$\{Pri\text{-}AgrK\ d,$
$Key\ (sesK\ (c * d * (s' + a * b))),\ Key\ (sesK\ (c * d * (s + a * b))),$
$\{|Key\ (sesK\ (c * d * (s + a * b))),\ Agent\ X,\ Number\ n,\ Number\ run|\}\} \cup$
$U) \in protocol\ |$

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

$C4$: $[\![(evsC4,\ S,\ A,\ U) \in protocol;$
$IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c;$
$ExtAgrK\ (S\ (User\ m,\ n,\ run)) = None;$
$Says\ n\ run\ 3\ X\ (User\ m)\ (pubAK\ f) \in set\ evsC4;$
$\{|Key\ (sesK\ (c * f)),\ Agent\ (User\ m),\ Number\ n,\ Number\ run|\} \in U]\!]$
$\Longrightarrow (Says\ n\ run\ 4\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ (c * f))\ (pubAK\ f))\ \#\ evsC4,$
$S\ ((User\ m,\ n,\ run) := S\ (User\ m,\ n,\ run)\ (|ExtAgrK := Some\ f|)),$
$A,\ U) \in protocol\ |$

$FC4$: $[\![(evsFC4,\ S,\ A,\ U) \in protocol;$
$NonceS\ (S\ (Card\ n,\ n,\ run)) = Some\ s;$
$IntMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ b;$
$ExtMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ a;$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d;$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e;$
$Crypt\ (sesK\ (d * e))\ (pubAK\ (d * (s + a * b)))$
$\in synth\ (analz\ (A \cup spies\ evsFC4))]\!]$
$\Longrightarrow (Says\ n\ run\ 4\ Spy\ (Card\ n)$
$(Crypt\ (sesK\ (d * e))\ (pubAK\ (d * (s + a * b))))\ \#\ evsFC4,$
$S,\ A,\ U) \in protocol\ |$

$R4$: $[\![(evsR4,\ S,\ A,\ U) \in protocol;$
$NonceS\ (S\ (Card\ n,\ n,\ run)) = Some\ s;$
$IntMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ b;$
$ExtMapK\ (S\ (Card\ n,\ n,\ run)) = Some\ a;$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d;$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e;$
$Says\ n\ run\ 4\ X\ (Card\ n)\ (Crypt\ (sesK\ (d * e))$
$(pubAK\ (d * (s + a * b)))) \in set\ evsR4]\!]$
$\Longrightarrow (Says\ n\ run\ 4\ (Card\ n)\ X\ (Crypt\ (sesK\ (d * e))$
$\{|pubAK\ e,\ Auth\text{-}Data\ (priAK\ n)\ b,\ pubAK\ (priAK\ n),$

$$
\begin{array}{l}
\quad\quad Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ (priAK\ n)))\}) \#\ evsR4,\\
\quad\quad S,\ A,\ U) \in protocol\ |
\end{array}
$$

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

$$
\begin{array}{l}
C5\colon\ \ [\![(evsC5,\ S,\ A,\ U) \in protocol;\\
\quad\quad NonceS\ (S\ (User\ m,\ n,\ run)) = Some\ s;\\
\quad\quad IntMapK\ (S\ (User\ m,\ n,\ run)) = Some\ a;\\
\quad\quad ExtMapK\ (S\ (User\ m,\ n,\ run)) = Some\ b;\\
\quad\quad IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c;\\
\quad\quad ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f;\\
\quad\quad Says\ n\ run\ 4\ X\ (User\ m)\ (Crypt\ (sesK\ (c * f))\\
\quad\quad\quad \{\!|pubAK\ (c * (s + a * b)),\ Auth\text{-}Data\ g\ b,\ pubAK\ g,\\
\quad\quad\quad\quad Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ g))|\!\}) \in set\ evsC5]\!]\\
\quad \Longrightarrow (Says\ n\ run\ 5\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m)) \#\\
evsC5,\\
\quad\quad S,\ A,\ U) \in protocol\ |
\end{array}
$$

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

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

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

$S,\ A,\ U) \in \mathit{protocol}$

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*:
$(\mathit{evs},\ S,\ A,\ U) \in \mathit{protocol} \Longrightarrow$
$(\mathit{NonceS}\ (S\ (X,\ n,\ \mathit{run})) = \mathit{None} \longrightarrow \mathit{IntMapK}\ (S\ (X,\ n,\ \mathit{run})) = \mathit{None}) \wedge$

$(IntMapK\ (S\ (X,\ n,\ run)) = None \longrightarrow ExtMapK\ (S\ (X,\ n,\ run)) = None) \wedge$
$(ExtMapK\ (S\ (X,\ n,\ run)) = None \longrightarrow IntAgrK\ (S\ (X,\ n,\ run)) = None) \wedge$
$(IntAgrK\ (S\ (X,\ n,\ run)) = None \longrightarrow ExtAgrK\ (S\ (X,\ n,\ run)) = None)$
**proof** (*erule protocol.induct*, *simp-all*)
**qed** (*rule-tac* [!] *impI*, *simp-all*)

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

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

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

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

**lemma** *pr-analz-used*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow A \subseteq U$
**by** (*erule protocol.induct*, *auto*)

**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** (*erule protocol.induct*, *subst parts-simp*, *simp*, *blast*, *simp*)
**qed** (*simp-all add*: *parts-simp-insert parts-auth-data parts-crypt parts-mpair*)

**lemma** *pr-key-analz*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow (Key\ K \in analz\ (A \cup spies\ evs)) = (Key\ K \in A)$
**proof** (*rule iffI*, *drule subsetD* [*OF analz-parts-subset*], *erule pr-key-parts-intro*, *assumption*)
**qed** (*rule subsetD* [*OF analz-subset*], *simp*)

**lemma** *pr-symk-used*:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow Key\ (symK\ n) \in U$
**by** (*erule protocol.induct*, *simp-all*)

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

*bad*)
**proof** (*simp add*: *pr-key-analz*, *erule protocol.induct*, *simp-all*, *rule-tac* [*2*] *impI*,
*rule-tac* [!] *iffI*, *simp-all*, *erule disjE*, *erule-tac* [*2*−*4*] *disjE*, *simp-all*)
**assume** *Key* (*symK n*) ∈ *Key* ‘ *symK* ‘ *bad*
**hence** ∃ *m* ∈ *bad*. *symK n* = *symK m*
**by** (*simp add*: *image-iff*)
**then obtain** *m* **where** *m* ∈ *bad* **and** *symK n* = *symK m* **..**
**thus** *n* ∈ *bad*
**by** (*rule symK-bad*)
**next**
**assume** *Key* (*symK n*) ∈ *Key* ‘ *range pubSK*
**thus** *n* ∈ *bad*
**by** (*auto*, *drule-tac sym*, *erule-tac notE* [*OF pubSK-symK*])
**next**
**assume** *Key* (*symK n*) ∈ *pubAK* ‘ *range priAK*
**thus** *n* ∈ *bad*
**by** *blast*
**next**
**fix** *evsR3 S A U s a b c d*
**assume** (*evsR3*, *S*, *A*, *U*) ∈ *protocol*
**hence** *Key* (*symK n*) ∈ *U*
**by** (*rule pr-symk-used*)
**moreover assume** *symK n* = *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** *simp*
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *n* ∈ *bad*
**by** *contradiction*
**next**
**fix** *evsFR3 S A U s a b c d*
**assume** (*evsFR3*, *S*, *A*, *U*) ∈ *protocol*
**hence** *Key* (*symK n*) ∈ *U*
**by** (*rule pr-symk-used*)
**moreover assume** *symK n* = *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** *simp*
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *n* ∈ *bad*
**by** *contradiction*
**qed**

**lemma** *pr-sesk-card* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* ⟶
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e* ⟶
*Key* (*sesK* (*d* ∗ *e*)) ∈ *U*
**proof** (*erule protocol.induct*, *simp-all*, (*rule impI*)+, *simp*)
**qed** (*subst* (*2*) *mult.commute*, *subst mult.assoc*, *simp*)

**lemma** *pr-sesk-user-1* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* ⟶
*ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f* ⟶
{|*Key* (*sesK* (*c* ∗ *f*)), *Agent* (*User m*), *Number n*, *Number run*|} ∈ *U*
**proof** (*erule protocol.induct*, *simp-all*, (*rule-tac* [!] *impI*)+, *simp-all*)
**fix** *evsC3 S A U m n run*
**assume** *A*: (*evsC3*, *S*, *A*, *U*) ∈ *protocol* **and**
*ExtMapK* (*S* (*User m*, *n*, *run*)) = *None*
**hence** *IntAgrK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-3*)
**with** *A* **have** *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-4*)
**moreover assume** *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f*
**ultimately show** {|*Key* (*sesK* (*c* ∗ *f*)), *Agent* (*User m*), *Number n*, *Number run*|}
∈ *U*
**by** *simp*
**qed**

**lemma** *pr-sesk-user-2* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
{|*Key* (*sesK K*), *Agent* (*User m*), *Number n*, *Number run*|} ∈ *U* ⟶
*Key* (*sesK K*) ∈ *U*
**by** (*erule protocol.induct*, *blast*, *simp-all*)

**lemma** *pr-auth-key-used*:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹ *Pri-AgrK* (*priAK n*) ∈ *U*
**by** (*erule protocol.induct*, *simp-all*)

**lemma** *pr-int-mapk-used* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b* ⟶
*Pri-AgrK b* ∈ *U*
**by** (*erule protocol.induct*, *simp-all*)

**lemma** *pr-valid-key-analz*:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹ *Key* (*pubSK X*) ∈ *analz* (*A* ∪ *spies evs*)
**by** (*simp add*: *pr-key-analz*, *erule protocol.induct*, *simp-all*)

**lemma** *pr-pri-agrk-parts* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*Pri-AgrK x* ∉ *U* ⟶
*Pri-AgrK x* ∉ *parts* (*A* ∪ *spies evs*)
**proof** (*induction arbitrary*: *x rule*: *protocol.induct*,
*simp-all add*: *parts-simp-insert parts-auth-data parts-crypt parts-mpair*,
*subst parts-simp*, *blast*, *blast*, *rule-tac* [!] *impI*)
**fix** *evsFR1 A U m s x*
**assume**
⋀*x*. *Pri-AgrK x* ∉ *U* ⟶ *Pri-AgrK x* ∉ *parts* (*A* ∪ *spies evsFR1*) **and**

```
   Pri-AgrK x ∉ U
  hence A: Pri-AgrK x ∉ parts (A ∪ spies evsFR1) ..
  assume B: Crypt (symK m) (Pri-AgrK s) ∈ synth (analz (A ∪ spies evsFR1))
  show x ≠ s
  proof
    assume x = s
    hence Crypt (symK m) (Pri-AgrK x) ∈ synth (analz (A ∪ spies evsFR1))
     using B by simp
    hence Crypt (symK m) (Pri-AgrK x) ∈ analz (A ∪ spies evsFR1) ∨
      Pri-AgrK x ∈ synth (analz (A ∪ spies evsFR1)) ∧
      Key (symK m) ∈ analz (A ∪ spies evsFR1)
      (is ?P ∨ ?Q)
     by (rule synth-crypt)
    moreover {
      assume ?P
      hence Crypt (symK m) (Pri-AgrK x) ∈ parts (A ∪ spies evsFR1)
       by (rule subsetD [OF analz-parts-subset])
      hence Pri-AgrK x ∈ parts (A ∪ spies evsFR1)
       by (rule parts.Body)
      hence False
       using A by contradiction
    }
    moreover {
      assume ?Q
      hence Pri-AgrK x ∈ synth (analz (A ∪ spies evsFR1)) ..
      hence Pri-AgrK x ∈ analz (A ∪ spies evsFR1)
       by (rule synth-simp-intro, simp)
      hence Pri-AgrK x ∈ parts (A ∪ spies evsFR1)
       by (rule subsetD [OF analz-parts-subset])
      hence False
       using A by contradiction
    }
    ultimately show False ..
  qed
next
  fix evsR4 S A U b n run x
  assume
    A: (evsR4, S, A, U) ∈ protocol and
    B: IntMapK (S (Card n, n, run)) = Some b and
    C: Pri-AgrK x ∉ U
  show x ≠ priAK n ∧ x ≠ b
  proof (rule conjI, rule-tac [!] notI)
    assume x = priAK n
    moreover have Pri-AgrK (priAK n) ∈ U
     using A by (rule pr-auth-key-used)
    ultimately have Pri-AgrK x ∈ U
     by simp
    thus False
     using C by contradiction
```

```
  next
    assume x = b
    moreover have Pri-AgrK b ∈ U
     using A and B by (rule pr-int-mapk-used)
    ultimately have Pri-AgrK x ∈ U
     by simp
    thus False
     using C by contradiction
  qed
next
  fix evsFR4 S A U s a b c f g x
  assume
    A: ⋀x. Pri-AgrK x ∉ U ⟶ Pri-AgrK x ∉ parts (A ∪ spies evsFR4) and
    B: (evsFR4, S, A, U) ∈ protocol and
    C: Crypt (sesK (c * f))
      {|pubAK (c * (s + a * b)), Auth-Data g b, pubAK g,
       Crypt (priSK CA) (Hash (pubAK g))|} ∈ synth (analz (A ∪ spies evsFR4))
      (is Crypt - ?M ∈ synth (analz ?A)) and
    D: Pri-AgrK x ∉ U
  show x ≠ g ∧ x ≠ b
  proof −
    have E: Pri-AgrK b ∈ U ∧ Pri-AgrK g ∈ U
    proof −
      have Crypt (sesK (c * f)) ?M ∈ analz ?A ∨
        ?M ∈ synth (analz ?A) ∧ Key (sesK (c * f)) ∈ analz ?A
       using C by (rule synth-crypt)
      moreover {
        assume Crypt (sesK (c * f)) ?M ∈ analz ?A
        hence Crypt (sesK (c * f)) ?M ∈ parts ?A
         by (rule subsetD [OF analz-parts-subset])
        hence ?M ∈ parts ?A
         by (rule parts.Body)
        hence {|Auth-Data g b, pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
          ∈ parts ?A
         by (rule parts.Snd)
        hence F: Auth-Data g b ∈ parts ?A
         by (rule parts.Fst)
        hence Pri-AgrK b ∈ parts ?A
         by (rule parts.Auth-Snd)
        moreover have Pri-AgrK g ∈ parts ?A
         using F by (rule parts.Auth-Fst)
        ultimately have Pri-AgrK b ∈ parts ?A ∧ Pri-AgrK g ∈ parts ?A ..
      }
      moreover {
        assume ?M ∈ synth (analz ?A) ∧
          Key (sesK (c * f)) ∈ analz ?A
        hence ?M ∈ synth (analz ?A) ..
        hence {|Auth-Data g b, pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
          ∈ synth (analz ?A)
```

```
    by (rule synth-analz-snd)
   hence Auth-Data g b ∈ synth (analz ?A)
    by (rule synth-analz-fst)
   hence Auth-Data g b ∈ analz ?A ∨
     Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b ∈ analz ?A
    by (rule synth-auth-data)
   moreover {
    assume Auth-Data g b ∈ analz ?A
    hence F: Auth-Data g b ∈ parts ?A
     by (rule subsetD [OF analz-parts-subset])
    hence Pri-AgrK b ∈ parts ?A
     by (rule parts.Auth-Snd)
    moreover have Pri-AgrK g ∈ parts ?A
     using F by (rule parts.Auth-Fst)
    ultimately have Pri-AgrK b ∈ parts ?A ∧ Pri-AgrK g ∈ parts ?A ..
   }
   moreover {
    assume F: Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b ∈ analz ?A
    hence Pri-AgrK b ∈ analz ?A ..
    hence Pri-AgrK b ∈ parts ?A
     by (rule subsetD [OF analz-parts-subset])
    moreover have Pri-AgrK g ∈ analz ?A
     using F ..
    hence Pri-AgrK g ∈ parts ?A
     by (rule subsetD [OF analz-parts-subset])
    ultimately have Pri-AgrK b ∈ parts ?A ∧ Pri-AgrK g ∈ parts ?A ..
   }
   ultimately have Pri-AgrK b ∈ parts ?A ∧ Pri-AgrK g ∈ parts ?A ..
  }
  ultimately have F: Pri-AgrK b ∈ parts ?A ∧ Pri-AgrK g ∈ parts ?A ..
  hence Pri-AgrK b ∈ parts ?A ..
  hence Pri-AgrK b ∈ U
   by (rule contrapos-pp, insert A, simp)
  moreover have Pri-AgrK g ∈ parts ?A
   using F ..
  hence Pri-AgrK g ∈ U
   by (rule contrapos-pp, insert A, simp)
  ultimately show ?thesis ..
qed
show ?thesis
proof (rule conjI, rule-tac [!] notI)
  assume x = g
  hence Pri-AgrK x ∈ U
   using E by simp
  thus False
   using D by contradiction
next
  assume x = b
  hence Pri-AgrK x ∈ U
```

```
      using E by simp
     thus False
      using D by contradiction
   qed
  qed
qed

lemma pr-pri-agrk-items:
 (evs, S, A, U) ∈ protocol ⟹
    Pri-AgrK x ∉ U ⟹
  items (insert (Pri-AgrK x) (A ∪ spies evs)) =
    insert (Pri-AgrK x) (items (A ∪ spies evs))
by (rule items-pri-agrk-out, rule pr-pri-agrk-parts)

lemma pr-auth-data-items:
 (evs, S, A, U) ∈ protocol ⟹
  Pri-AgrK (priAK n) ∉ items (A ∪ spies evs) ∧
  (IntMapK (S (Card n, n, run)) = Some b ⟶
    Pri-AgrK b ∉ items (A ∪ spies evs))
proof (induction arbitrary: n run b rule: protocol.induct,
 simp-all add: items-simp-insert-2 items-crypt items-mpair pr-pri-agrk-items,
 subst items-simp, blast+)
  fix evsR1 S A U n' run' s n run b
  assume
    A: (evsR1, S, A, U) ∈ protocol and
    B: Pri-AgrK s ∉ U
  show
   (n = n' ∧ run = run' ⟶
      priAK n' ≠ s ∧ (IntMapK (S (Card n', n', run')) = Some b ⟶ b ≠ s)) ∧
    ((n = n' ⟶ run ≠ run') ⟶
      priAK n ≠ s ∧ (IntMapK (S (Card n, n, run)) = Some b ⟶ b ≠ s))
  proof (rule conjI, rule-tac [!] impI, rule-tac [!] conjI, rule-tac [2] impI,
   rule-tac [4] impI, rule-tac [!] notI)
    have Pri-AgrK (priAK n) ∈ U
     using A by (rule pr-auth-key-used)
    moreover assume priAK n = s
    ultimately have Pri-AgrK s ∈ U
     by simp
    thus False
     using B by contradiction
  next
    assume IntMapK (S (Card n, n, run)) = Some b
    with A have Pri-AgrK b ∈ U
     by (rule pr-int-mapk-used)
    moreover assume b = s
    ultimately have Pri-AgrK s ∈ U
     by simp
    thus False
     using B by contradiction
```

```
  next
    have Pri-AgrK (priAK n') ∈ U
     using A by (rule pr-auth-key-used)
    moreover assume priAK n' = s
    ultimately have Pri-AgrK s ∈ U
     by simp
    thus False
     using B by contradiction
  next
    assume IntMapK (S (Card n', n', run')) = Some b
    with A have Pri-AgrK b ∈ U
     by (rule pr-int-mapk-used)
    moreover assume b = s
    ultimately have Pri-AgrK s ∈ U
     by simp
    thus False
     using B by contradiction
  qed
next
  fix evsFR1 A m s n run b and S :: state
  assume A: ⋀n run b. Pri-AgrK (priAK n) ∉ items (A ∪ spies evsFR1) ∧
    (IntMapK (S (Card n, n, run)) = Some b ⟶
      Pri-AgrK b ∉ items (A ∪ spies evsFR1))
  assume Crypt (symK m) (Pri-AgrK s) ∈ synth (analz (A ∪ spies evsFR1))
  hence Crypt (symK m) (Pri-AgrK s) ∈ analz (A ∪ spies evsFR1) ∨
    Pri-AgrK s ∈ synth (analz (A ∪ spies evsFR1)) ∧
    Key (symK m) ∈ analz (A ∪ spies evsFR1)
    (is ?P ∨ ?Q)
   by (rule synth-crypt)
  moreover {
    assume ?P
    hence Crypt (symK m) (Pri-AgrK s) ∈ items (A ∪ spies evsFR1)
     by (rule subsetD [OF analz-items-subset])
    hence Pri-AgrK s ∈ items (A ∪ spies evsFR1)
     by (rule items.Body)
  }
  moreover {
    assume ?Q
    hence Pri-AgrK s ∈ synth (analz (A ∪ spies evsFR1)) ..
    hence Pri-AgrK s ∈ analz (A ∪ spies evsFR1)
     by (rule synth-simp-intro, simp)
    hence Pri-AgrK s ∈ items (A ∪ spies evsFR1)
     by (rule subsetD [OF analz-items-subset])
  }
  ultimately have B: Pri-AgrK s ∈ items (A ∪ spies evsFR1) ..
  show Pri-AgrK (priAK n) ∉ items (insert (Pri-AgrK s) (A ∪ spies evsFR1)) ∧
    (IntMapK (S (Card n, n, run)) = Some b ⟶
      Pri-AgrK b ∉ items (insert (Pri-AgrK s) (A ∪ spies evsFR1)))
   by (simp add: items-simp-insert-1 [OF B] A)
```

**next**
**fix** *evsC2 S A U a n run b* **and** *m* :: *nat*
**assume**
*A*: (*evsC2*, *S*, *A*, *U*) ∈ *protocol* **and**
*B*: *Pri-AgrK a* ∉ *U*
**show** *m* = *0* ⟶ *priAK n* ≠ *a* ∧ (*IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b* ⟶ *b* ≠ *a*)
**proof** (*rule impI*, *rule conjI*, *rule-tac* [*2*] *impI*, *rule-tac* [!] *notI*)
**have** *Pri-AgrK* (*priAK n*) ∈ *U*
**using** *A* **by** (*rule pr-auth-key-used*)
**moreover assume** *priAK n* = *a*
**ultimately have** *Pri-AgrK a* ∈ *U*
**by** *simp*
**thus** *False*
**using** *B* **by** *contradiction*
**next**
**assume** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b*
**with** *A* **have** *Pri-AgrK b* ∈ *U*
**by** (*rule pr-int-mapk-used*)
**moreover assume** *b* = *a*
**ultimately have** *Pri-AgrK a* ∈ *U*
**by** *simp*
**thus** *False*
**using** *B* **by** *contradiction*
**qed**
**next**
**fix** *evsR2 S A U b′ n′ run′ b* **and** *n* :: *nat* **and** *run* :: *nat*
**assume**
*A*: (*evsR2*, *S*, *A*, *U*) ∈ *protocol* **and**
*B*: *Pri-AgrK b′* ∉ *U*
**show** *n* = *n′* ∧ *run* = *run′* ⟶ *b′* = *b* ⟶ *Pri-AgrK b* ∉ *items* (*A* ∪ *spies evsR2*)
**proof** ((*rule impI*)+, *drule sym*, *simp*)
**qed** (*rule contra-subsetD* [*OF items-parts-subset*], *rule pr-pri-agrk-parts* [*OF A B*])
**next**
**fix** *evsC3 S A U c n run b* **and** *m* :: *nat*
**assume**
*A*: (*evsC3*, *S*, *A*, *U*) ∈ *protocol* **and**
*B*: *Pri-AgrK c* ∉ *U*
**show** *m* = *0* ⟶ *priAK n* ≠ *c* ∧ (*IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b* ⟶ *b* ≠ *c*)
**proof** (*rule impI*, *rule conjI*, *rule-tac* [*2*] *impI*, *rule-tac* [!] *notI*)
**have** *Pri-AgrK* (*priAK n*) ∈ *U*
**using** *A* **by** (*rule pr-auth-key-used*)
**moreover assume** *priAK n* = *c*
**ultimately have** *Pri-AgrK c* ∈ *U*
**by** *simp*
**thus** *False*
**using** *B* **by** *contradiction*

**next**
**assume** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b*
**with** *A* **have** *Pri-AgrK b* ∈ *U*
**by** (*rule pr-int-mapk-used*)
**moreover assume** *b* = *c*
**ultimately have** *Pri-AgrK c* ∈ *U*
**by** *simp*
**thus** *False*
**using** *B* **by** *contradiction*
**qed**
**next**
**fix** *evsR3 A n′ run′ s b′ c n run b* **and** *S* :: *state* **and** *s′* :: *pri-agrk*
**assume**
*A*: ⋀*n run b*. *Pri-AgrK* (*priAK n*) ∉ *items* (*A* ∪ *spies evsR3*) ∧
(*IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b* ⟶
*Pri-AgrK b* ∉ *items* (*A* ∪ *spies evsR3*)) **and**
*B*: *IntMapK* (*S* (*Card n′*, *n′*, *run′*)) = *Some b′*
**show**
(*s′* = *s* ∧ *Pri-AgrK c* ∈ *analz* (*A* ∪ *spies evsR3*) ⟶
*n* = *n′* ∧ *run* = *run′* ⟶ *b′* = *b* ⟶
*Pri-AgrK b* ∉ *items* (*A* ∪ *spies evsR3*)) ∧
((*s′* = *s* ⟶ *Pri-AgrK c* ∉ *analz* (*A* ∪ *spies evsR3*)) ⟶
*n* = *n′* ∧ *run* = *run′* ⟶ *b′* = *b* ⟶
*Pri-AgrK b* ∉ *items* (*A* ∪ *spies evsR3*))
**proof** (*rule conjI*, (*rule-tac* [!] *impI*)+)
**have** *Pri-AgrK* (*priAK n′*) ∉ *items* (*A* ∪ *spies evsR3*) ∧
(*IntMapK* (*S* (*Card n′*, *n′*, *run′*)) = *Some b′* ⟶
*Pri-AgrK b′* ∉ *items* (*A* ∪ *spies evsR3*))
**using** *A* **.**
**hence** *Pri-AgrK b′* ∉ *items* (*A* ∪ *spies evsR3*)
**using** *B* **by** *simp*
**moreover assume** *b′* = *b*
**ultimately show** *Pri-AgrK b* ∉ *items* (*A* ∪ *spies evsR3*)
**by** *simp*
**next**
**have** *Pri-AgrK* (*priAK n′*) ∉ *items* (*A* ∪ *spies evsR3*) ∧
(*IntMapK* (*S* (*Card n′*, *n′*, *run′*)) = *Some b′* ⟶
*Pri-AgrK b′* ∉ *items* (*A* ∪ *spies evsR3*))
**using** *A* **.**
**hence** *Pri-AgrK b′* ∉ *items* (*A* ∪ *spies evsR3*)
**using** *B* **by** *simp*
**moreover assume** *b′* = *b*
**ultimately show** *Pri-AgrK b* ∉ *items* (*A* ∪ *spies evsR3*)
**by** *simp*
**qed**
**next**
**fix** *evsR4 A n′ run′ b′ n run b* **and** *S* :: *state*
**let** *?M* = {|*pubAK* (*priAK n′*), *Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK n′*)))|}
**assume**

```
    A: ⋀n run b. Pri-AgrK (priAK n) ∉ items (A ∪ spies evsR4) ∧
      (IntMapK (S (Card n, n, run)) = Some b ⟶
        Pri-AgrK b ∉ items (A ∪ spies evsR4)) and
    B: IntMapK (S (Card n', n', run')) = Some b'
  show
   Pri-AgrK (priAK n) ∉ items (insert (Auth-Data (priAK n') b')
     (insert ?M (A ∪ spies evsR4))) ∧
    (IntMapK (S (Card n, n, run)) = Some b ⟶
      Pri-AgrK b ∉ items (insert (Auth-Data (priAK n') b')
        (insert ?M (A ∪ spies evsR4))))
  proof (subst (1 2) insert-commute, simp add: items-mpair,
   subst (1 3) insert-commute, simp add: items-simp-insert-2,
   subst (1 2) insert-commute, simp add: items-crypt items-simp-insert-2)
    have C: Pri-AgrK (priAK n') ∉ items (A ∪ spies evsR4) ∧
      (IntMapK (S (Card n', n', run')) = Some b' ⟶
        Pri-AgrK b' ∉ items (A ∪ spies evsR4))
     using A .
    hence Pri-AgrK (priAK n') ∉ items (A ∪ spies evsR4) ..
    moreover have Pri-AgrK b' ∉ items (A ∪ spies evsR4)
     using B and C by simp
    ultimately show
     Pri-AgrK (priAK n) ∉ items (insert (Auth-Data (priAK n') b')
       (A ∪ spies evsR4)) ∧
      (IntMapK (S (Card n, n, run)) = Some b ⟶
        Pri-AgrK b ∉ items (insert (Auth-Data (priAK n') b')
          (A ∪ spies evsR4)))
     by (simp add: items-auth-data-out A)
  qed
next
  fix evsFR4 A s a b' c f g n run b and S :: state
  let ?M = {|pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
  assume
    A: ⋀n run b. Pri-AgrK (priAK n) ∉ items (A ∪ spies evsFR4) ∧
      (IntMapK (S (Card n, n, run)) = Some b ⟶
        Pri-AgrK b ∉ items (A ∪ spies evsFR4)) and
    B: Crypt (sesK (c * f)) {|pubAK (c * (s + a * b')), Auth-Data g b', ?M|}
      ∈ synth (analz (A ∪ spies evsFR4))
    (is Crypt - ?M' ∈ synth (analz ?A))
  have C: Pri-AgrK b' ∈ items ?A ∨ Pri-AgrK g ∈ items ?A ⟶
    Pri-AgrK b' ∈ items ?A ∧ Pri-AgrK g ∈ items ?A
    (is ?P ⟶ ?Q)
  proof
    assume ?P
    have Crypt (sesK (c * f)) ?M' ∈ analz ?A ∨
      ?M' ∈ synth (analz ?A) ∧ Key (sesK (c * f)) ∈ analz ?A
     using B by (rule synth-crypt)
    moreover {
      assume Crypt (sesK (c * f)) ?M' ∈ analz ?A
      hence Crypt (sesK (c * f)) ?M' ∈ items ?A
```

```
  by (rule subsetD [OF analz-items-subset])
 hence ?M' ∈ items ?A
  by (rule items.Body)
 hence {|Auth-Data g b', pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
   ∈ items ?A
  by (rule items.Snd)
 hence D: Auth-Data g b' ∈ items ?A
  by (rule items.Fst)
 have ?Q
 proof (rule disjE [OF ‹?P›])
   assume Pri-AgrK b' ∈ items ?A
   moreover from this have Pri-AgrK g ∈ items ?A
    by (rule items.Auth-Fst [OF D])
   ultimately show ?Q ..
 next
   assume Pri-AgrK g ∈ items ?A
   moreover from this have Pri-AgrK b' ∈ items ?A
    by (rule items.Auth-Snd [OF D])
   ultimately show ?Q
    by simp
 qed
}
moreover {
 assume ?M' ∈ synth (analz ?A) ∧ Key (sesK (c * f)) ∈ analz ?A
 hence ?M' ∈ synth (analz ?A) ..
 hence {|Auth-Data g b', pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
   ∈ synth (analz ?A)
  by (rule synth-analz-snd)
 hence Auth-Data g b' ∈ synth (analz ?A)
  by (rule synth-analz-fst)
 hence Auth-Data g b' ∈ analz ?A ∨
   Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b' ∈ analz ?A
  by (rule synth-auth-data)
 moreover {
   assume Auth-Data g b' ∈ analz ?A
   hence D: Auth-Data g b' ∈ items ?A
    by (rule subsetD [OF analz-items-subset])
   have ?Q
   proof (rule disjE [OF ‹?P›])
     assume Pri-AgrK b' ∈ items ?A
     moreover from this have Pri-AgrK g ∈ items ?A
      by (rule items.Auth-Fst [OF D])
     ultimately show ?Q ..
   next
     assume Pri-AgrK g ∈ items ?A
     moreover from this have Pri-AgrK b' ∈ items ?A
      by (rule items.Auth-Snd [OF D])
     ultimately show ?Q
      by simp
```

```
      qed
    }
    moreover {
      assume D: Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b' ∈ analz ?A
      hence Pri-AgrK b' ∈ analz ?A ..
      hence Pri-AgrK b' ∈ items ?A
       by (rule subsetD [OF analz-items-subset])
      moreover have Pri-AgrK g ∈ analz ?A
       using D ..
      hence Pri-AgrK g ∈ items ?A
       by (rule subsetD [OF analz-items-subset])
      ultimately have ?Q ..
    }
    ultimately have ?Q ..
  }
  ultimately show ?Q ..
 qed
 show
  Pri-AgrK (priAK n) ∉ items (insert (Auth-Data g b')
     (insert ?M (A ∪ spies evsFR4))) ∧
   (IntMapK (S (Card n, n, run)) = Some b ⟶
     Pri-AgrK b ∉ items (insert (Auth-Data g b')
      (insert ?M (A ∪ spies evsFR4))))
 proof (subst (1 2) insert-commute, simp add: items-mpair,
  subst (1 3) insert-commute, simp add: items-simp-insert-2,
   subst (1 2) insert-commute, simp add: items-crypt items-simp-insert-2, cases
?P)
   case True
   with C have ?Q ..
   thus
    Pri-AgrK (priAK n) ∉ items (insert (Auth-Data g b')
       (A ∪ spies evsFR4)) ∧
     (IntMapK (S (Card n, n, run)) = Some b ⟶
       Pri-AgrK b ∉ items (insert (Auth-Data g b')
        (A ∪ spies evsFR4)))
    by (simp add: items-auth-data-in items-simp-insert-1 A)
 next
   case False
   thus
    Pri-AgrK (priAK n) ∉ items (insert (Auth-Data g b')
       (A ∪ spies evsFR4)) ∧
     (IntMapK (S (Card n, n, run)) = Some b ⟶
       Pri-AgrK b ∉ items (insert (Auth-Data g b')
        (A ∪ spies evsFR4)))
    by (simp add: items-auth-data-out A)
 qed
qed

lemma pr-auth-key-analz:
```

(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹ *Pri-AgrK* (*priAK n*) ∉ *analz* (*A* ∪ *spies evs*)
**proof** (*rule contra-subsetD* [*OF analz-items-subset*], *drule pr-auth-data-items*)
**qed** (*erule conjE*)

**lemma** *pr-int-mapk-analz*:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b* ⟹
*Pri-AgrK b* ∉ *analz* (*A* ∪ *spies evs*)
**proof** (*rule contra-subsetD* [*OF analz-items-subset*], *drule pr-auth-data-items*)
**qed** (*erule conjE*, *rule mp*)

**lemma** *pr-key-set-unused* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*H* ∪ *A* ∪ *spies evs*) = *H* ∪ *analz* (*A* ∪ *spies evs*)
**proof** (*induction arbitrary*: *H rule*: *protocol.induct*, *simp-all add*: *analz-simp-insert-2*,
*rule impI*, (*subst analz-simp*, *blast*)+, *simp*)
**fix** *evsR1 S A U n s H*
**assume**
*A*: ⋀*H*. *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*H* ∪ *A* ∪ *spies evsR1*) = *H* ∪ *analz* (*A* ∪ *spies evsR1*) **and**
*B*: (*evsR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *Pri-AgrK s* ∉ *U*
**let**
*?B* = *H* ∪ *A* ∪ *spies evsR1* **and**
*?C* = *A* ∪ *spies evsR1*
**show**
(*n* ∈ *bad* ⟶
*H* ⊆ *range Key* ∪ *range Pri-AgrK* − *insert* (*Pri-AgrK s*) *U* ⟶
*analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) (*insert* (*Pri-AgrK s*) *?B*)) =
*H* ∪ *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) (*insert* (*Pri-AgrK s*) *?C*)))
∧
(*n* ∉ *bad* ⟶
*H* ⊆ *range Key* ∪ *range Pri-AgrK* − *insert* (*Pri-AgrK s*) *U* ⟶
*analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) *?B*) =
*H* ∪ *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) *?C*))
(**is** (- ⟶ - ⟶ *?T*) ∧ (- ⟶ - ⟶ *?T'*))
**proof** (*rule conjI*, (*rule-tac* [!] *impI*)+)
**assume** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *insert* (*Pri-AgrK s*) *U*
**hence** *insert* (*Pri-AgrK s*) *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**using** *C* **by** *blast*
**with** *A* **have** *analz* (*insert* (*Pri-AgrK s*) *H* ∪ *A* ∪ *spies evsR1*) =
*insert* (*Pri-AgrK s*) *H* ∪ *analz* (*A* ∪ *spies evsR1*) **..**
**hence** *analz* (*insert* (*Pri-AgrK s*) *?B*) = *H* ∪ *insert* (*Pri-AgrK s*) (*analz ?C*)
**by** *simp*
**moreover have** {*Pri-AgrK s*} ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**using** *C* **by** *simp*
**with** *A* **have** *analz* ({*Pri-AgrK s*} ∪ *A* ∪ *spies evsR1*) =
{*Pri-AgrK s*} ∪ *analz* (*A* ∪ *spies evsR1*) **..**

**hence** *insert* (*Pri-AgrK s*) (*analz ?C*) = *analz* (*insert* (*Pri-AgrK s*) *?C*)
**by** *simp*
**ultimately have** *D*: *analz* (*insert* (*Pri-AgrK s*) *?B*) =
*H* ∪ *analz* (*insert* (*Pri-AgrK s*) *?C*)
**by** *simp*
**assume** *n* ∈ *bad*
**hence** *E*: *Key* (*invK* (*symK n*)) ∈ *analz ?C*
**using** *B* **by** (*simp add*: *pr-symk-analz invK-symK*)
**have** *Key* (*invK* (*symK n*)) ∈ *analz* (*insert* (*Pri-AgrK s*) *?B*)
**by** (*rule subsetD* [*OF - E*], *rule analz-mono*, *blast*)
**hence** *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) (*insert* (*Pri-AgrK s*) *?B*))
=
*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) (*analz* (*insert* (*Pri-AgrK s*) *?B*))
**by** (*simp add*: *analz-crypt-in*)
**moreover have** *Key* (*invK* (*symK n*)) ∈ *analz* (*insert* (*Pri-AgrK s*) *?C*)
**by** (*rule subsetD* [*OF - E*], *rule analz-mono*, *blast*)
**hence** *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) (*insert* (*Pri-AgrK s*) *?C*))
=
*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) (*analz* (*insert* (*Pri-AgrK s*) *?C*))
**by** (*simp add*: *analz-crypt-in*)
**ultimately show** *?T*
**using** *D* **by** *simp*
**next**
**assume** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *insert* (*Pri-AgrK s*) *U*
**hence** *D*: *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**by** *blast*
**with** *A* **have** *E*: *analz ?B* = *H* ∪ *analz ?C* **..**
**moreover assume** *n* ∉ *bad*
**hence** *F*: *Key* (*invK* (*symK n*)) ∉ *analz ?C*
**using** *B* **by** (*simp add*: *pr-symk-analz invK-symK*)
**ultimately have** *Key* (*invK* (*symK n*)) ∉ *analz ?B*
**proof** (*simp add*: *invK-symK*, *insert pr-symk-used* [*OF B*, *of n*])
**qed** (*rule notI*, *drule subsetD* [*OF D*], *simp*)
**hence** *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) *?B*) =
*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) (*analz ?B*)
**by** (*simp add*: *analz-crypt-out*)
**moreover have** *H* ∪ *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) *?C*) =
*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*)) (*H* ∪ *analz ?C*)
**using** *F* **by** (*simp add*: *analz-crypt-out*)
**ultimately show** *?T′*
**using** *E* **by** *simp*
**qed**
**next**
**fix** *evsFR1 S A U m s H*
**assume**
*A*: ⋀*H*. *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*H* ∪ *A* ∪ *spies evsFR1*) = *H* ∪ *analz* (*A* ∪ *spies evsFR1*) **and**
*B*: (*evsFR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *Crypt* (*symK m*) (*Pri-AgrK s*) ∈ *synth* (*analz* (*A* ∪ *spies evsFR1*))

**let**
*?B* = *H* ∪ *A* ∪ *spies evsFR1* **and**
*?C* = *A* ∪ *spies evsFR1*
**show** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) *?B*) =
*H* ∪ *analz* (*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) *?C*)
(**is** - ⟶ *?T*)
**proof** (*rule impI*, *cases Key* (*invK* (*symK m*)) ∈ *analz ?C*)
**case** *True*
**assume** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**with** *A* **have** *analz ?B* = *H* ∪ *analz ?C* **..**
**moreover have** *Pri-AgrK s* ∈ *analz ?C*
**proof** (*insert synth-crypt* [*OF C*], *erule disjE*, *erule-tac* [*2*] *conjE*)
**assume** *Crypt* (*symK m*) (*Pri-AgrK s*) ∈ *analz ?C*
**thus** *?thesis*
**using** *True* **by** (*rule analz.Decrypt*)
**next**
**assume** *Pri-AgrK s* ∈ *synth* (*analz ?C*)
**thus** *?thesis*
**by** (*rule synth-simp-intro*, *simp*)
**qed**
**moreover from** *this* **have** *Pri-AgrK s* ∈ *analz ?B*
**by** (*rule rev-subsetD*, *rule-tac analz-mono*, *blast*)
**ultimately have** *D*: *analz* (*insert* (*Pri-AgrK s*) *?B*) =
*H* ∪ *analz* (*insert* (*Pri-AgrK s*) *?C*)
**by** (*simp add*: *analz-simp-insert-1*)
**have** *Key* (*invK* (*symK m*)) ∈ *analz ?B*
**by** (*rule subsetD* [*OF* - *True*], *rule analz-mono*, *blast*)
**hence** *analz* (*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) *?B*) =
*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) (*analz* (*insert* (*Pri-AgrK s*) *?B*))
**by** (*simp add*: *analz-crypt-in*)
**moreover have** *analz* (*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) *?C*) =
*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) (*analz* (*insert* (*Pri-AgrK s*) *?C*))
**using** *True* **by** (*simp add*: *analz-crypt-in*)
**ultimately show** *?T*
**using** *D* **by** *simp*
**next**
**case** *False*
**moreover assume** *D*: *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**with** *A* **have** *E*: *analz ?B* = *H* ∪ *analz ?C* **..**
**ultimately have** *Key* (*invK* (*symK m*)) ∉ *analz ?B*
**proof** (*simp add*: *invK-symK*, *insert pr-symk-used* [*OF B*, *of m*])
**qed** (*rule notI*, *drule subsetD* [*OF D*], *simp*)
**hence** *analz* (*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) *?B*) =
*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) (*analz ?B*)
**by** (*simp add*: *analz-crypt-out*)
**moreover have** *H* ∪ *analz* (*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) *?C*) =
*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*)) (*H* ∪ *analz ?C*)
**using** *False* **by** (*simp add*: *analz-crypt-out*)

```
    ultimately show ?T
     using E by simp
  qed
next
  fix evsC2 S A U a H and m :: nat
  assume
    A: ⋀H. H ⊆ range Key ∪ range Pri-AgrK − U ⟶
      analz (H ∪ A ∪ spies evsC2) = H ∪ analz (A ∪ spies evsC2) and
    B: Pri-AgrK a ∉ U
  let
    ?B = H ∪ A ∪ spies evsC2 and
    ?C = A ∪ spies evsC2
  show
   (m = 0 ⟶
      H ⊆ range Key ∪ range Pri-AgrK − insert (Pri-AgrK a) U ⟶
     insert (pubAK a) (analz (insert (Pri-AgrK a) ?B)) =
     insert (pubAK a) (H ∪ analz (insert (Pri-AgrK a) ?C))) ∧
    (0 < m ⟶
      H ⊆ range Key ∪ range Pri-AgrK − insert (Pri-AgrK a) U ⟶
     insert (pubAK a) (analz ?B) =
     insert (pubAK a) (H ∪ analz ?C))
    (is (- ⟶ - ⟶ ?T) ∧ (- ⟶ - ⟶ ?T'))
  proof (rule conjI, (rule-tac [!] impI)+)
    assume H ⊆ range Key ∪ range Pri-AgrK − insert (Pri-AgrK a) U
    hence insert (Pri-AgrK a) H ⊆ range Key ∪ range Pri-AgrK − U
     using B by blast
    with A have analz (insert (Pri-AgrK a) H ∪ A ∪ spies evsC2) =
      insert (Pri-AgrK a) H ∪ analz (A ∪ spies evsC2) ..
    hence analz (insert (Pri-AgrK a) ?B) = H ∪ insert (Pri-AgrK a) (analz ?C)
     by simp
    moreover have {Pri-AgrK a} ⊆ range Key ∪ range Pri-AgrK − U
     using B by simp
    with A have analz ({Pri-AgrK a} ∪ A ∪ spies evsC2) =
      {Pri-AgrK a} ∪ analz (A ∪ spies evsC2) ..
    hence insert (Pri-AgrK a) (analz ?C) = analz (insert (Pri-AgrK a) ?C)
     by simp
    ultimately have analz (insert (Pri-AgrK a) ?B) =
      H ∪ analz (insert (Pri-AgrK a) ?C)
     by simp
    thus ?T
     by simp
  next
    assume H ⊆ range Key ∪ range Pri-AgrK − insert (Pri-AgrK a) U
    hence H ⊆ range Key ∪ range Pri-AgrK − U
     by blast
    with A have analz ?B = H ∪ analz ?C ..
    thus ?T'
     by simp
  qed
```

**next**
**fix** *evsR2 S A U b H*
**assume** *A*: $\bigwedge H.\ H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U \longrightarrow$
*analz* $(H \cup A \cup spies\ evsR2) = H \cup analz\ (A \cup spies\ evsR2)$
**let**
$?B = H \cup A \cup spies\ evsR2$ **and**
$?C = A \cup spies\ evsR2$
**show** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - insert\ (Pri\text{-}AgrK\ b)\ U \longrightarrow$
*insert* $(pubAK\ b)\ (analz\ ?B) = insert\ (pubAK\ b)\ (H \cup analz\ ?C)$
(**is** - $\longrightarrow$ *?T*)
**proof**
**assume** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - insert\ (Pri\text{-}AgrK\ b)\ U$
**hence** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**by** *blast*
**with** *A* **have** $analz\ ?B = H \cup analz\ ?C$ **..**
**thus** *?T*
**by** *simp*
**qed**
**next**
**fix** *evsC3 S A U s a b c H* **and** $m :: nat$
**assume**
*A*: $\bigwedge H.\ H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U \longrightarrow$
*analz* $(H \cup A \cup spies\ evsC3) = H \cup analz\ (A \cup spies\ evsC3)$ **and**
*B*: $Pri\text{-}AgrK\ c \notin U$
**let**
$?B = H \cup A \cup spies\ evsC3$ **and**
$?C = A \cup spies\ evsC3$
**show**
$(m = 0 \longrightarrow$
$H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - insert\ (Pri\text{-}AgrK\ c)\ U \longrightarrow$
$insert\ (pubAK\ (c * (s + a * b)))\ (analz\ (insert\ (Pri\text{-}AgrK\ c)\ ?B)) =$
$insert\ (pubAK\ (c * (s + a * b)))\ (H \cup analz\ (insert\ (Pri\text{-}AgrK\ c)\ ?C))) \wedge$
$(0 < m \longrightarrow$
$H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - insert\ (Pri\text{-}AgrK\ c)\ U \longrightarrow$
$insert\ (pubAK\ (c * (s + a * b)))\ (analz\ ?B) =$
$insert\ (pubAK\ (c * (s + a * b)))\ (H \cup analz\ ?C))$
(**is** (- $\longrightarrow$ - $\longrightarrow$ *?T*) $\wedge$ (- $\longrightarrow$ - $\longrightarrow$ *?T'*))
**proof** (*rule conjI*, (*rule-tac* [!] *impI*)+)
**assume** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - insert\ (Pri\text{-}AgrK\ c)\ U$
**hence** $insert\ (Pri\text{-}AgrK\ c)\ H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**using** *B* **by** *blast*
**with** *A* **have** $analz\ (insert\ (Pri\text{-}AgrK\ c)\ H \cup A \cup spies\ evsC3) =$
$insert\ (Pri\text{-}AgrK\ c)\ H \cup analz\ (A \cup spies\ evsC3)$ **..**
**hence** $analz\ (insert\ (Pri\text{-}AgrK\ c)\ ?B) = H \cup insert\ (Pri\text{-}AgrK\ c)\ (analz\ ?C)$
**by** *simp*
**moreover have** $\{Pri\text{-}AgrK\ c\} \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**using** *B* **by** *simp*
**with** *A* **have** $analz\ (\{Pri\text{-}AgrK\ c\} \cup A \cup spies\ evsC3) =$
$\{Pri\text{-}AgrK\ c\} \cup analz\ (A \cup spies\ evsC3)$ **..**

```
    hence insert (Pri-AgrK c) (analz ?C) = analz (insert (Pri-AgrK c) ?C)
     by simp
    ultimately have analz (insert (Pri-AgrK c) ?B) =
      H ∪ analz (insert (Pri-AgrK c) ?C)
     by simp
    thus ?T
     by simp
  next
    assume H ⊆ range Key ∪ range Pri-AgrK − insert (Pri-AgrK c) U
    hence H ⊆ range Key ∪ range Pri-AgrK − U
     by blast
    with A have analz ?B = H ∪ analz ?C ..
    thus ?T'
     by simp
  qed
next
  fix evsR3 S A U n run s s' a b c d X H
  assume
    A: ⋀H. H ⊆ range Key ∪ range Pri-AgrK − U ⟶
      analz (H ∪ A ∪ spies evsR3) = H ∪ analz (A ∪ spies evsR3) and
    B: Key (sesK (c * d * (s + a * b))) ∉ U
      (is Key ?K ∉ -)
  let
    ?B = H ∪ A ∪ spies evsR3 and
    ?C = A ∪ spies evsR3 and
    ?K' = sesK (c * d * (s' + a * b))
  show
   (s' = s ∧ Pri-AgrK c ∈ analz (A ∪ spies evsR3) ⟶
      H ⊆ range Key ∪ range Pri-AgrK − insert (Pri-AgrK d)
       (insert (Key ?K) (insert {|Key ?K, Agent X, Number n, Number run|} U))
⟶
    insert (pubAK (d * (s + a * b))) (analz (insert (Key ?K) ?B)) =
    insert (pubAK (d * (s + a * b))) (H ∪ analz (insert (Key ?K) ?C))) ∧
   ((s' = s ⟶ Pri-AgrK c ∉ analz (A ∪ spies evsR3)) ⟶
      H ⊆ range Key ∪ range Pri-AgrK − insert (Pri-AgrK d) (insert (Key ?K')
       (insert (Key ?K) (insert {|Key ?K, Agent X, Number n, Number run|} U)))
⟶
    insert (pubAK (d * (s + a * b))) (analz ?B) =
    insert (pubAK (d * (s + a * b))) (H ∪ analz ?C))
   (is (- ⟶ - ⟶ ?T) ∧ (- ⟶ - ⟶ ?T'))
  proof (rule conjI, (rule-tac [!] impI)+)
    assume H ⊆ range Key ∪ range Pri-AgrK − insert (Pri-AgrK d)
      (insert (Key ?K) (insert {|Key ?K, Agent X, Number n, Number run|} U))
    hence insert (Key ?K) H ⊆ range Key ∪ range Pri-AgrK − U
     using B by blast
    with A have analz (insert (Key ?K) H ∪ A ∪ spies evsR3) =
      insert (Key ?K) H ∪ analz (A ∪ spies evsR3) ..
    hence analz (insert (Key ?K) ?B) = H ∪ insert (Key ?K) (analz ?C)
     by simp
```

**moreover have** $\{Key\ ?K\} \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**using** $B$ **by** *simp*
**with** $A$ **have** $analz\ (\{Key\ ?K\} \cup A \cup spies\ evsR3) =$
$\{Key\ ?K\} \cup analz\ (A \cup spies\ evsR3)$ **..**
**hence** $insert\ (Key\ ?K)\ (analz\ ?C) = analz\ (insert\ (Key\ ?K)\ ?C)$
**by** *simp*
**ultimately have** $analz\ (insert\ (Key\ ?K)\ ?B) = H \cup analz\ (insert\ (Key\ ?K)\ ?C)$
**by** *simp*
**thus** *?T*
**by** *simp*
**next**
**assume** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - insert\ (Pri\text{-}AgrK\ d)\ (insert\ (Key\ ?K')$
$(insert\ (Key\ ?K)\ (insert\ \{|Key\ ?K, Agent\ X, Number\ n, Number\ run|\}\ U)))$
**hence** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**by** *blast*
**with** $A$ **have** $analz\ ?B = H \cup analz\ ?C$ **..**
**thus** *?T'*
**by** *simp*
**qed**
**next**
**fix** *evsFR3 S A U m n run s a b c d H*
**assume**
$A$: $\bigwedge H.\ H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U \longrightarrow$
$analz\ (H \cup A \cup spies\ evsFR3) = H \cup analz\ (A \cup spies\ evsFR3)$ **and**
$B$: $Key\ (sesK\ (c * d * (s + a * b))) \notin U$
(**is** $Key\ ?K \notin$ -)
**let**
$?B = H \cup A \cup spies\ evsFR3$ **and**
$?C = A \cup spies\ evsFR3$
**show**
$H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - insert\ (Key\ ?K)$
$(insert\ \{|Key\ ?K, Agent\ (User\ m), Number\ n, Number\ run|\}\ U) \longrightarrow$
$insert\ (pubAK\ (d * (s + a * b)))\ (analz\ (insert\ (Key\ ?K)\ ?B)) =$
$insert\ (pubAK\ (d * (s + a * b)))\ (H \cup analz\ (insert\ (Key\ ?K)\ ?C))$
(**is** - $\longrightarrow$ *?T*)
**proof**
**assume** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - insert\ (Key\ ?K)$
$(insert\ \{|Key\ ?K, Agent\ (User\ m), Number\ n, Number\ run|\}\ U)$
**hence** $insert\ (Key\ ?K)\ H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**using** $B$ **by** *blast*
**with** $A$ **have** $analz\ (insert\ (Key\ ?K)\ H \cup A \cup spies\ evsFR3) =$
$insert\ (Key\ ?K)\ H \cup analz\ (A \cup spies\ evsFR3)$ **..**
**hence** $analz\ (insert\ (Key\ ?K)\ ?B) = H \cup insert\ (Key\ ?K)\ (analz\ ?C)$
**by** *simp*
**moreover have** $\{Key\ ?K\} \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**using** $B$ **by** *simp*
**with** $A$ **have** $analz\ (\{Key\ ?K\} \cup A \cup spies\ evsFR3) =$

```
    {Key ?K} ∪ analz (A ∪ spies evsFR3) ..
   hence insert (Key ?K) (analz ?C) = analz (insert (Key ?K) ?C)
    by simp
   ultimately have analz (insert (Key ?K) ?B) = H ∪ analz (insert (Key ?K)
?C)
    by simp
   thus ?T
    by simp
  qed
next
  fix evsC4 S A U m n run c f H
  assume
   A: ⋀H. H ⊆ range Key ∪ range Pri-AgrK − U ⟶
     analz (H ∪ A ∪ spies evsC4) = H ∪ analz (A ∪ spies evsC4) and
   B: (evsC4, S, A, U) ∈ protocol and
   C: {|Key (sesK (c ∗ f)), Agent (User m), Number n, Number run|} ∈ U
  let
   ?B = H ∪ A ∪ spies evsC4 and
   ?C = A ∪ spies evsC4
  show H ⊆ range Key ∪ range Pri-AgrK − U ⟶
   analz (insert (Crypt (sesK (c ∗ f)) (pubAK f)) ?B) =
   H ∪ analz (insert (Crypt (sesK (c ∗ f)) (pubAK f)) ?C)
   (is - ⟶ ?T)
  proof (rule impI, cases Key (invK (sesK (c ∗ f))) ∈ analz ?C)
   case True
   assume H ⊆ range Key ∪ range Pri-AgrK − U
   with A have D: analz ?B = H ∪ analz ?C ..
   have Key (invK (sesK (c ∗ f))) ∈ analz ?B
    by (rule subsetD [OF - True], rule analz-mono, blast)
   hence analz (insert (Crypt (sesK (c ∗ f)) (pubAK f)) ?B) =
     insert (Crypt (sesK (c ∗ f)) (pubAK f)) (insert (pubAK f) (analz ?B))
    by (simp add: analz-crypt-in analz-simp-insert-2)
   moreover have H ∪ analz (insert (Crypt (sesK (c ∗ f)) (pubAK f)) ?C) =
     insert (Crypt (sesK (c ∗ f)) (pubAK f)) (insert (pubAK f) (H ∪ analz ?C))
    using True by (simp add: analz-crypt-in analz-simp-insert-2)
   ultimately show ?T
    using D by simp
  next
   case False
   moreover assume D: H ⊆ range Key ∪ range Pri-AgrK − U
   with A have E: analz ?B = H ∪ analz ?C ..
   ultimately have Key (invK (sesK (c ∗ f))) ∉ analz ?B
   proof (simp add: invK-sesK, insert pr-sesk-user-2 [OF B C])
   qed (rule notI, drule subsetD [OF D], simp)
   hence analz (insert (Crypt (sesK (c ∗ f)) (pubAK f)) ?B) =
     insert (Crypt (sesK (c ∗ f)) (pubAK f)) (analz ?B)
    by (simp add: analz-crypt-out)
   moreover have H ∪ analz (insert (Crypt (sesK (c ∗ f)) (pubAK f)) ?C) =
     insert (Crypt (sesK (c ∗ f)) (pubAK f)) (H ∪ analz ?C)
```

**using** *False* **by** (*simp add*: *analz-crypt-out*)
**ultimately show** *?T*
**using** *E* **by** *simp*
**qed**
**next**
**fix** *evsFC4 S A U n run s a b d e H*
**assume**
*A*: ⋀*H*. *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*H* ∪ *A* ∪ *spies evsFC4*) = *H* ∪ *analz* (*A* ∪ *spies evsFC4*) **and**
*B*: (*evsFC4*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*D*: *ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**let**
*?B* = *H* ∪ *A* ∪ *spies evsFC4* **and**
*?C* = *A* ∪ *spies evsFC4* **and**
*?f* = *d* ∗ (*s* + *a* ∗ *b*)
**show** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) *?B*) =
*H* ∪ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) *?C*)
(**is** - ⟶ *?T*)
**proof** (*rule impI*, *cases Key* (*invK* (*sesK* (*d* ∗ *e*))) ∈ *analz ?C*)
**case** *True*
**assume** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**with** *A* **have** *E*: *analz ?B* = *H* ∪ *analz ?C* **..**
**have** *Key* (*invK* (*sesK* (*d* ∗ *e*))) ∈ *analz ?B*
**by** (*rule subsetD* [*OF - True*], *rule analz-mono*, *blast*)
**hence** *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) *?B*) =
*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) (*insert* (*pubAK ?f*) (*analz ?B*))
**by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**moreover have** *H* ∪ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) *?C*) =
*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) (*insert* (*pubAK ?f*) (*H* ∪ *analz ?C*))
**using** *True* **by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**ultimately show** *?T*
**using** *E* **by** *simp*
**next**
**case** *False*
**moreover assume** *E*: *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**with** *A* **have** *F*: *analz ?B* = *H* ∪ *analz ?C* **..**
**ultimately have** *Key* (*invK* (*sesK* (*d* ∗ *e*))) ∉ *analz ?B*
**proof** (*simp add*: *invK-sesK*, *insert pr-sesk-card* [*OF B C D*])
**qed** (*rule notI*, *drule subsetD* [*OF E*], *simp*)
**hence** *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) *?B*) =
*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) (*analz ?B*)
**by** (*simp add*: *analz-crypt-out*)
**moreover have** *H* ∪ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) *?C*) =
*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK ?f*)) (*H* ∪ *analz ?C*)
**using** *False* **by** (*simp add*: *analz-crypt-out*)
**ultimately show** *?T*

**using** *F* **by** *simp*
**qed**
**next**
**fix** *evsR4 S A U n run b d e H*
**let**
*?B* = *H* ∪ *A* ∪ *spies evsR4* **and**
*?C* = *A* ∪ *spies evsR4* **and**
*?H* = *Hash* (*pubAK* (*priAK n*)) **and**
*?M* = {|*pubAK* (*priAK n*), *Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK n*)))|} **and**
*?M′* = {|*pubAK e*, *Auth-Data* (*priAK n*) *b*, *pubAK* (*priAK n*),
*Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK n*)))|}
**assume**
*A*: ⋀*H*. *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*H* ∪ *A* ∪ *spies evsR4*) = *H* ∪ *analz* (*A* ∪ *spies evsR4*) **and**
*B*: (*evsR4*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b* **and**
*D*: *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*E*: *ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**show** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) *?M′*) *?B*) =
*H* ∪ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) *?M′*) *?C*)
(**is** - ⟶ *?T*)
**proof**
**assume** *F*: *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**with** *A* **have** *G*: *analz ?B* = *H* ∪ *analz ?C* **..**
**have** *H*: *Key* (*pubSK CA*) ∈ *analz ?C*
**using** *B* **by** (*rule pr-valid-key-analz*)
**hence** *I*: *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?C*) =
{*Crypt* (*priSK CA*) *?H*, *?H*} ∪ *analz ?C*
**by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**have** *Key* (*pubSK CA*) ∈ *analz ?B*
**by** (*rule subsetD* [*OF - H*], *rule analz-mono*, *blast*)
**hence** *J*: *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?B*) =
{*Crypt* (*priSK CA*) *?H*, *?H*} ∪ *analz ?B*
**by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**have** *K*: *Pri-AgrK* (*priAK n*) ∉ *analz ?C*
**using** *B* **by** (*rule pr-auth-key-analz*)
**hence** *L*: *Pri-AgrK* (*priAK n*) ∉ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?C*)
**using** *I* **by** *simp*
**have** *M*: *Pri-AgrK b* ∉ *analz ?C*
**using** *B* **and** *C* **by** (*rule pr-int-mapk-analz*)
**hence** *N*: *Pri-AgrK b* ∉ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?C*)
**using** *I* **by** *simp*
**have** *Pri-AgrK* (*priAK n*) ∈ *U*
**using** *B* **by** (*rule pr-auth-key-used*)
**hence** *Pri-AgrK* (*priAK n*) ∉ *H*
**using** *F* **by** *blast*
**hence** *O*: *Pri-AgrK* (*priAK n*) ∉ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?B*)
**using** *G* **and** *J* **and** *K* **by** *simp*

```
    have Pri-AgrK b ∈ U
     using B and C by (rule pr-int-mapk-used)
    hence Pri-AgrK b ∉ H
     using F by blast
    hence P: Pri-AgrK b ∉ analz (insert (Crypt (priSK CA) ?H) ?B)
     using G and J and M by simp
    show ?T
    proof (cases Key (invK (sesK (d * e))) ∈ analz ?C)
      case True
      have Q: Key (invK (sesK (d * e))) ∈ analz ?B
       by (rule subsetD [OF - True], rule analz-mono, blast)
      show ?T
      proof (simp add: analz-crypt-in analz-mpair analz-simp-insert-2 True Q,
       rule equalityI, (rule-tac [!] insert-mono)+)
        show analz (insert (Auth-Data (priAK n) b) (insert ?M ?B)) ⊆
          H ∪ analz (insert (Auth-Data (priAK n) b) (insert ?M ?C))
      proof (subst (1 2) insert-commute, simp add: analz-mpair analz-simp-insert-2
         del: Un-insert-right, subst (1 3) insert-commute,
         subst analz-auth-data-out [OF O P], subst analz-auth-data-out [OF L N])
        qed (auto simp add: G I J)
      next
        show H ∪ analz (insert (Auth-Data (priAK n) b) (insert ?M ?C)) ⊆
          analz (insert (Auth-Data (priAK n) b) (insert ?M ?B))
      proof (subst (1 2) insert-commute, simp add: analz-mpair analz-simp-insert-2
         del: Un-insert-right Un-subset-iff semilattice-sup-class.sup.bounded-iff,
         subst (1 3) insert-commute, subst analz-auth-data-out [OF L N],
         subst analz-auth-data-out [OF O P])
        qed (auto simp add: G I J)
      qed
    next
      case False
      hence Key (invK (sesK (d * e))) ∉ analz ?B
      proof (simp add: invK-sesK G, insert pr-sesk-card [OF B D E])
      qed (rule notI, drule subsetD [OF F], simp)
      hence analz (insert (Crypt (sesK (d * e)) ?M') ?B) =
        insert (Crypt (sesK (d * e)) ?M') (analz ?B)
       by (simp add: analz-crypt-out)
      moreover have H ∪ analz (insert (Crypt (sesK (d * e)) ?M') ?C) =
        insert (Crypt (sesK (d * e)) ?M') (H ∪ analz ?C)
       using False by (simp add: analz-crypt-out)
      ultimately show ?T
       using G by simp
    qed
  qed
next
  fix evsFR4 S A U m n run s a b c f g H
  let
    ?B = H ∪ A ∪ spies evsFR4 and
    ?C = A ∪ spies evsFR4 and
```

```
  ?H = Hash (pubAK g) and
  ?M = {|pubAK g, Crypt (priSK CA) (Hash (pubAK g))|} and
  ?M' = {|pubAK (c * (s + a * b)), Auth-Data g b, pubAK g,
    Crypt (priSK CA) (Hash (pubAK g))|}
assume
  A: ⋀H. H ⊆ range Key ∪ range Pri-AgrK − U ⟶
    analz (H ∪ A ∪ spies evsFR4) = H ∪ analz (A ∪ spies evsFR4) and
  B: (evsFR4, S, A, U) ∈ protocol and
  C: IntAgrK (S (User m, n, run)) = Some c and
  D: ExtAgrK (S (User m, n, run)) = Some f and
  E: Crypt (sesK (c * f)) ?M' ∈ synth (analz ?C)
have F:
 Key (invK (sesK (c * f))) ∈ analz ?C ⟶
    Pri-AgrK b ∈ analz ?C ∨ Pri-AgrK g ∈ analz ?C ⟶
  Pri-AgrK b ∈ analz ?C ∧ Pri-AgrK g ∈ analz ?C
  (is ?P ⟶ ?Q ⟶ ?R)
proof (rule impI)+
  assume ?P and ?Q
  have Crypt (sesK (c * f)) ?M' ∈ analz ?C ∨
    ?M' ∈ synth (analz ?C) ∧ Key (sesK (c * f)) ∈ analz ?C
   using E by (rule synth-crypt)
  moreover {
    assume Crypt (sesK (c * f)) ?M' ∈ analz ?C
    hence ?M' ∈ analz ?C
     using ‹?P› by (rule analz.Decrypt)
    hence {|Auth-Data g b, pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
      ∈ analz ?C
     by (rule analz.Snd)
    hence G: Auth-Data g b ∈ analz ?C
     by (rule analz.Fst)
    have ?R
    proof (rule disjE [OF ‹?Q›])
      assume Pri-AgrK b ∈ analz ?C
      moreover from this have Pri-AgrK g ∈ analz ?C
       by (rule analz.Auth-Fst [OF G])
      ultimately show ?R ..
    next
      assume Pri-AgrK g ∈ analz ?C
      moreover from this have Pri-AgrK b ∈ analz ?C
       by (rule analz.Auth-Snd [OF G])
      ultimately show ?R
       by simp
    qed
  }
  moreover {
    assume ?M' ∈ synth (analz ?C) ∧ Key (sesK (c * f)) ∈ analz ?C
    hence ?M' ∈ synth (analz ?C) ..
    hence {|Auth-Data g b, pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
      ∈ synth (analz ?C)
```

```
    by (rule synth-analz-snd)
   hence Auth-Data g b ∈ synth (analz ?C)
    by (rule synth-analz-fst)
   hence Auth-Data g b ∈ analz ?C ∨
     Pri-AgrK g ∈ analz ?C ∧ Pri-AgrK b ∈ analz ?C
    by (rule synth-auth-data)
   moreover {
    assume G: Auth-Data g b ∈ analz ?C
    have ?R
    proof (rule disjE [OF ‹?Q›])
     assume Pri-AgrK b ∈ analz ?C
     moreover from this have Pri-AgrK g ∈ analz ?C
      by (rule analz.Auth-Fst [OF G])
     ultimately show ?R ..
    next
     assume Pri-AgrK g ∈ analz ?C
     moreover from this have Pri-AgrK b ∈ analz ?C
      by (rule analz.Auth-Snd [OF G])
     ultimately show ?R
      by simp
    qed
   }
   moreover {
    assume Pri-AgrK g ∈ analz ?C ∧ Pri-AgrK b ∈ analz ?C
    hence ?R
     by simp
   }
   ultimately have ?R ..
  }
  ultimately show ?R ..
qed
show H ⊆ range Key ∪ range Pri-AgrK − U ⟶
  analz (insert (Crypt (sesK (c ∗ f)) ?M′) ?B) =
  H ∪ analz (insert (Crypt (sesK (c ∗ f)) ?M′) ?C)
  (is - ⟶ ?T)
proof
  assume G: H ⊆ range Key ∪ range Pri-AgrK − U
  with A have H: analz ?B = H ∪ analz ?C ..
  have I: Key (pubSK CA) ∈ analz ?C
   using B by (rule pr-valid-key-analz)
  hence J: analz (insert (Crypt (priSK CA) ?H) ?C) =
    {Crypt (priSK CA) ?H, ?H} ∪ analz ?C
   by (simp add: analz-crypt-in analz-simp-insert-2)
  have Key (pubSK CA) ∈ analz ?B
   by (rule subsetD [OF - I], rule analz-mono, blast)
  hence K: analz (insert (Crypt (priSK CA) ?H) ?B) =
    {Crypt (priSK CA) ?H, ?H} ∪ analz ?B
   by (simp add: analz-crypt-in analz-simp-insert-2)
  show ?T
```

**proof** (*cases Key* (*invK* (*sesK* (*c* ∗ *f*))) ∈ *analz ?C*,
*cases Pri-AgrK g* ∈ *analz ?C* ∨ *Pri-AgrK b* ∈ *analz ?C*, *simp-all*)
**assume** *L*: *Key* (*invK* (*sesK* (*c* ∗ *f*))) ∈ *analz ?C*
**have** *M*: *Key* (*invK* (*sesK* (*c* ∗ *f*))) ∈ *analz ?B*
**by** (*rule subsetD* [*OF - L*], *rule analz-mono*, *blast*)
**assume** *N*: *Pri-AgrK g* ∈ *analz ?C* ∨ *Pri-AgrK b* ∈ *analz ?C*
**hence** *O*: *Pri-AgrK g* ∈ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?C*) ∨
*Pri-AgrK b* ∈ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?C*)
**using** *J* **by** *simp*
**have** *Pri-AgrK g* ∈ *analz ?B* ∨ *Pri-AgrK b* ∈ *analz ?B*
**using** *H* **and** *N* **by** *blast*
**hence** *P*: *Pri-AgrK g* ∈ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?B*) ∨
*Pri-AgrK b* ∈ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?B*)
**using** *K* **by** *simp*
**have** *Q*: *Pri-AgrK b* ∈ *analz ?C* ∧ *Pri-AgrK g* ∈ *analz ?C*
**using** *F* **and** *L* **and** *N* **by** *blast*
**hence** *Pri-AgrK g* ∈ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?C*)
**using** *J* **by** *simp*
**hence** *R*: *Pri-AgrK g* ∈ *analz* (*insert* (*Pri-AgrK b*)
(*insert* (*Crypt* (*priSK CA*) *?H*) *?C*))
**by** (*rule rev-subsetD*, *rule-tac analz-mono*, *blast*)
**have** *S*: *Pri-AgrK b* ∈ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?C*)
**using** *J* **and** *Q* **by** *simp*
**have** *T*: *Pri-AgrK b* ∈ *analz ?B* ∧ *Pri-AgrK g* ∈ *analz ?B*
**using** *H* **and** *Q* **by** *simp*
**hence** *Pri-AgrK g* ∈ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?B*)
**using** *K* **by** *simp*
**hence** *U*: *Pri-AgrK g* ∈ *analz* (*insert* (*Pri-AgrK b*)
(*insert* (*Crypt* (*priSK CA*) *?H*) *?B*))
**by** (*rule rev-subsetD*, *rule-tac analz-mono*, *blast*)
**have** *V*: *Pri-AgrK b* ∈ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?B*)
**using** *K* **and** *T* **by** *simp*
**show** *?T*
**proof** (*simp add*: *analz-crypt-in analz-mpair analz-simp-insert-2 L M*,
*rule equalityI*, (*rule-tac* [!] *insert-mono*)+)
**show** *analz* (*insert* (*Auth-Data g b*) (*insert ?M ?B*)) ⊆
*H* ∪ *analz* (*insert* (*Auth-Data g b*) (*insert ?M ?C*))
**proof** (*subst* (*1 2*) *insert-commute*, *simp add*: *analz-mpair analz-simp-insert-2*
*del*: *Un-insert-right*, *subst* (*1 3*) *insert-commute*,
*subst analz-auth-data-in* [*OF P*], *subst analz-auth-data-in* [*OF O*],
*simp del*: *Un-insert-right*)
**show**
*analz* (*insert* (*Pri-AgrK g*) (*insert* (*Pri-AgrK b*)
(*insert* (*Crypt* (*priSK CA*) *?H*) *?B*))) ⊆
*H* ∪ *insert ?M* (*insert* (*pubAK g*) (*insert* (*Auth-Data g b*)
(*analz* (*insert* (*Pri-AgrK g*) (*insert* (*Pri-AgrK b*)
(*insert* (*Crypt* (*priSK CA*) *?H*) *?C*))))))
**proof** (*subst analz-simp-insert-1* [*OF U*], *subst analz-simp-insert-1* [*OF V*],

```
        subst analz-simp-insert-1 [OF R], subst analz-simp-insert-1 [OF S])
      qed (auto simp add: H J K)
    qed
  next
    show H ∪ analz (insert (Auth-Data g b) (insert ?M ?C)) ⊆
      analz (insert (Auth-Data g b) (insert ?M ?B))
  proof (subst (1 2) insert-commute, simp add: analz-mpair analz-simp-insert-2
      del: Un-insert-right Un-subset-iff semilattice-sup-class.sup.bounded-iff,
      subst (2 4) insert-commute, subst analz-auth-data-in [OF O],
      subst analz-auth-data-in [OF P], simp only: Un-insert-left Un-empty-left)
      show
       H ∪ insert ?M (insert (pubAK g) (insert (Auth-Data g b)
         (analz (insert (Pri-AgrK g) (insert (Pri-AgrK b)
           (insert (Crypt (priSK CA) ?H) ?C)))))) ⊆
        insert ?M (insert (pubAK g) (insert (Auth-Data g b)
         (analz (insert (Pri-AgrK g) (insert (Pri-AgrK b)
           (insert (Crypt (priSK CA) ?H) ?B))))))
       proof (subst analz-simp-insert-1 [OF R], subst analz-simp-insert-1 [OF
S],
        subst analz-simp-insert-1 [OF U], subst analz-simp-insert-1 [OF V])
      qed (auto simp add: H J K)
    qed
  qed
next
  assume L: Key (invK (sesK (c * f))) ∈ analz ?C
  have M: Key (invK (sesK (c * f))) ∈ analz ?B
   by (rule subsetD [OF - L], rule analz-mono, blast)
  assume N: Pri-AgrK g ∉ analz ?C ∧ Pri-AgrK b ∉ analz ?C
  hence O: Pri-AgrK g ∉ analz (insert (Crypt (priSK CA) ?H) ?C)
   using J by simp
  have P: Pri-AgrK b ∉ analz (insert (Crypt (priSK CA) ?H) ?C)
   using J and N by simp
  have Q: Pri-AgrK g ∈ U ∧ Pri-AgrK b ∈ U
  proof −
    have Crypt (sesK (c * f)) ?M' ∈ analz ?C ∨
      ?M' ∈ synth (analz ?C) ∧ Key (sesK (c * f)) ∈ analz ?C
     using E by (rule synth-crypt)
    moreover {
      assume Crypt (sesK (c * f)) ?M' ∈ analz ?C
      hence Crypt (sesK (c * f)) ?M' ∈ parts ?C
       by (rule subsetD [OF analz-parts-subset])
      hence ?M' ∈ parts ?C
       by (rule parts.Body)
      hence {|Auth-Data g b, pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
        ∈ parts ?C
       by (rule parts.Snd)
      hence R: Auth-Data g b ∈ parts ?C
       by (rule parts.Fst)
      hence Pri-AgrK g ∈ parts ?C
```

```
   by (rule parts.Auth-Fst)
  hence Pri-AgrK g ∈ U
   by (rule contrapos-pp, rule-tac pr-pri-agrk-parts [OF B])
  moreover have Pri-AgrK b ∈ parts ?C
   using R by (rule parts.Auth-Snd)
  hence Pri-AgrK b ∈ U
   by (rule contrapos-pp, rule-tac pr-pri-agrk-parts [OF B])
  ultimately have ?thesis ..
}
moreover {
  assume ?M' ∈ synth (analz ?C) ∧
    Key (sesK (c ∗ f)) ∈ analz ?C
  hence ?M' ∈ synth (analz ?C) ..
  hence {|Auth-Data g b, pubAK g,
    Crypt (priSK CA) (Hash (pubAK g))|} ∈ synth (analz ?C)
   by (rule synth-analz-snd)
  hence Auth-Data g b ∈ synth (analz ?C)
   by (rule synth-analz-fst)
  hence Auth-Data g b ∈ analz ?C ∨
    Pri-AgrK g ∈ analz ?C ∧ Pri-AgrK b ∈ analz ?C
   by (rule synth-auth-data)
  moreover {
    assume Auth-Data g b ∈ analz ?C
    hence R: Auth-Data g b ∈ parts ?C
     by (rule subsetD [OF analz-parts-subset])
    hence Pri-AgrK g ∈ parts ?C
     by (rule parts.Auth-Fst)
    hence Pri-AgrK g ∈ U
     by (rule contrapos-pp, rule-tac pr-pri-agrk-parts [OF B])
    moreover have Pri-AgrK b ∈ parts ?C
     using R by (rule parts.Auth-Snd)
    hence Pri-AgrK b ∈ U
     by (rule contrapos-pp, rule-tac pr-pri-agrk-parts [OF B])
    ultimately have ?thesis ..
  }
  moreover {
    assume R: Pri-AgrK g ∈ analz ?C ∧ Pri-AgrK b ∈ analz ?C
    hence Pri-AgrK g ∈ analz ?C ..
    hence Pri-AgrK g ∈ parts ?C
     by (rule subsetD [OF analz-parts-subset])
    hence Pri-AgrK g ∈ U
     by (rule contrapos-pp, rule-tac pr-pri-agrk-parts [OF B])
    moreover have Pri-AgrK b ∈ analz ?C
     using R ..
    hence Pri-AgrK b ∈ parts ?C
     by (rule subsetD [OF analz-parts-subset])
    hence Pri-AgrK b ∈ U
     by (rule contrapos-pp, rule-tac pr-pri-agrk-parts [OF B])
    ultimately have ?thesis ..
```

```
        }
        ultimately have ?thesis ..
       }
       ultimately show ?thesis ..
      qed
      have R: Pri-AgrK g ∉ analz ?B ∧ Pri-AgrK b ∉ analz ?B
      proof (simp add: H N, rule conjI, rule-tac [!] notI, drule-tac [!] subsetD [OF
G])
      qed (simp-all add: Q)
      hence S: Pri-AgrK g ∉ analz (insert (Crypt (priSK CA) ?H) ?B)
       using K by simp
      have T: Pri-AgrK b ∉ analz (insert (Crypt (priSK CA) ?H) ?B)
       using K and R by simp
      show ?T
      proof (simp add: analz-crypt-in analz-mpair analz-simp-insert-2 L M,
       rule equalityI, (rule-tac [!] insert-mono)+)
        show analz (insert (Auth-Data g b) (insert ?M ?B)) ⊆
          H ∪ analz (insert (Auth-Data g b) (insert ?M ?C))
      proof (subst (1 2) insert-commute, simp add: analz-mpair analz-simp-insert-2
          del: Un-insert-right, subst (1 3) insert-commute,
          subst analz-auth-data-out [OF S T], subst analz-auth-data-out [OF O P])
        qed (auto simp add: H J K)
      next
        show H ∪ analz (insert (Auth-Data g b) (insert ?M ?C)) ⊆
          analz (insert (Auth-Data g b) (insert ?M ?B))
      proof (subst (1 2) insert-commute, simp add: analz-mpair analz-simp-insert-2
          del: Un-insert-right Un-subset-iff semilattice-sup-class.sup.bounded-iff,
          subst (2 4) insert-commute, subst analz-auth-data-out [OF O P],
          subst analz-auth-data-out [OF S T])
        qed (simp add: H J K)
      qed
     next
      assume L: Key (invK (sesK (c ∗ f))) ∉ analz ?C
      hence Key (invK (sesK (c ∗ f))) ∉ analz ?B
      proof (simp add: invK-sesK, insert pr-sesk-user-1 [OF B C D,
       THEN pr-sesk-user-2 [OF B]])
      qed (rule notI, simp add: H, drule subsetD [OF G], simp)
      hence analz (insert (Crypt (sesK (c ∗ f)) ?M′) ?B) =
        insert (Crypt (sesK (c ∗ f)) ?M′) (analz ?B)
       by (simp add: analz-crypt-out)
      moreover have H ∪ analz (insert (Crypt (sesK (c ∗ f)) ?M′) ?C) =
        insert (Crypt (sesK (c ∗ f)) ?M′) (H ∪ analz ?C)
       using L by (simp add: analz-crypt-out)
      ultimately show ?T
       using H by simp
     qed
    qed
  next
    fix evsC5 S A U m n run c f H
```

**assume**
*A*: $\bigwedge H.\ H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U \longrightarrow$
$analz\ (H \cup A \cup spies\ evsC5) = H \cup analz\ (A \cup spies\ evsC5)$ **and**
*B*: $(evsC5,\ S,\ A,\ U) \in protocol$ **and**
*C*: $IntAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ c$ **and**
*D*: $ExtAgrK\ (S\ (User\ m,\ n,\ run)) = Some\ f$
**let**
$?B = H \cup A \cup spies\ evsC5$ **and**
$?C = A \cup spies\ evsC5$
**show** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U \longrightarrow$
$analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ ?B) =$
$H \cup analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ ?C)$
(**is** - $\longrightarrow$ *?T*)
**proof** $(rule\ impI,\ cases\ Key\ (invK\ (sesK\ (c * f))) \in analz\ ?C)$
**case** *True*
**assume** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**with** *A* **have** *E*: $analz\ ?B = H \cup analz\ ?C$ **..**
**have** $Key\ (invK\ (sesK\ (c * f))) \in analz\ ?B$
**by** $(rule\ subsetD\ [OF\ -\ True],\ rule\ analz\text{-}mono,\ blast)$
**hence** $analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ ?B) =$
$insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ (insert\ (Passwd\ m)\ (analz\ ?B))$
**by** $(simp\ add:\ analz\text{-}crypt\text{-}in\ analz\text{-}simp\text{-}insert\text{-}2)$
**moreover have** $H \cup analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ ?C) =$
$insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ (insert\ (Passwd\ m)\ (H \cup analz\ ?C))$
**using** *True* **by** $(simp\ add:\ analz\text{-}crypt\text{-}in\ analz\text{-}simp\text{-}insert\text{-}2)$
**ultimately show** *?T*
**using** *E* **by** *simp*
**next**
**case** *False*
**moreover assume** *E*: $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**with** *A* **have** *F*: $analz\ ?B = H \cup analz\ ?C$ **..**
**ultimately have** $Key\ (invK\ (sesK\ (c * f))) \notin analz\ ?B$
**proof** $(simp\ add:\ invK\text{-}sesK,\ insert\ pr\text{-}sesk\text{-}user\text{-}1\ [OF\ B\ C\ D,$
$THEN\ pr\text{-}sesk\text{-}user\text{-}2\ [OF\ B]])$
**qed** $(rule\ notI,\ drule\ subsetD\ [OF\ E],\ simp)$
**hence** $analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ ?B) =$
$insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ (analz\ ?B)$
**by** $(simp\ add:\ analz\text{-}crypt\text{-}out)$
**moreover have** $H \cup analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ ?C) =$
$insert\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))\ (H \cup analz\ ?C)$
**using** *False* **by** $(simp\ add:\ analz\text{-}crypt\text{-}out)$
**ultimately show** *?T*
**using** *F* **by** *simp*
**qed**
**next**
**fix** *evsFC5 S A U n run d e H*
**assume**
*A*: $\bigwedge H.\ H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U \longrightarrow$

*analz* $(H \cup A \cup$ *spies evsFC5*$) = H \cup$ *analz* $(A \cup$ *spies evsFC5*$)$ **and**
*B*: $($*evsFC5*, *S*, *A*, *U*$) \in$ *protocol* **and**
*C*: *IntAgrK* $($*S* $($*Card n*, *n*, *run*$)) =$ *Some d* **and**
*D*: *ExtAgrK* $($*S* $($*Card n*, *n*, *run*$)) =$ *Some e*
**let**
*?B* $= H \cup A \cup$ *spies evsFC5* **and**
*?C* $= A \cup$ *spies evsFC5*
**show** $H \subseteq$ *range Key* $\cup$ *range Pri-AgrK* $- U \longrightarrow$
*analz* $($*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ *?B*$) =$
$H \cup$ *analz* $($*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ *?C*$)$
(**is** - $\longrightarrow$ *?T*)
**proof** $($*rule impI*, *cases Key* $($*invK* $($*sesK* $(d * e))) \in$ *analz ?C*$)$
**case** *True*
**assume** $H \subseteq$ *range Key* $\cup$ *range Pri-AgrK* $- U$
**with** *A* **have** *E*: *analz ?B* $= H \cup$ *analz ?C* **..**
**have** *Key* $($*invK* $($*sesK* $(d * e))) \in$ *analz ?B*
**by** $($*rule subsetD* $[$*OF - True*$]$, *rule analz-mono*, *blast*$)$
**hence** *analz* $($*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ *?B*$) =$
*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ $($*insert* $($*Passwd n*$)$ $($*analz ?B*$))$
**by** $($*simp add*: *analz-crypt-in analz-simp-insert-2*$)$
**moreover have** $H \cup$ *analz* $($*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ *?C*$) =$
*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ $($*insert* $($*Passwd n*$)$ $(H \cup$ *analz ?C*$))$
**using** *True* **by** $($*simp add*: *analz-crypt-in analz-simp-insert-2*$)$
**ultimately show** *?T*
**using** *E* **by** *simp*
**next**
**case** *False*
**moreover assume** *E*: $H \subseteq$ *range Key* $\cup$ *range Pri-AgrK* $- U$
**with** *A* **have** *F*: *analz ?B* $= H \cup$ *analz ?C* **..**
**ultimately have** *Key* $($*invK* $($*sesK* $(d * e))) \notin$ *analz ?B*
**proof** $($*simp add*: *invK-sesK*, *insert pr-sesk-card* $[$*OF B C D*$]$$)$
**qed** $($*rule notI*, *drule subsetD* $[$*OF E*$]$, *simp*$)$
**hence** *analz* $($*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ *?B*$) =$
*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ $($*analz ?B*$)$
**by** $($*simp add*: *analz-crypt-out*$)$
**moreover have** $H \cup$ *analz* $($*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ *?C*$) =$
*insert* $($*Crypt* $($*sesK* $(d * e))$ $($*Passwd n*$))$ $(H \cup$ *analz ?C*$)$
**using** *False* **by** $($*simp add*: *analz-crypt-out*$)$
**ultimately show** *?T*
**using** *F* **by** *simp*
**qed**
**next**
**fix** *evsR5 S A U n run d e H*
**assume**
*A*: $\bigwedge H.$ $H \subseteq$ *range Key* $\cup$ *range Pri-AgrK* $- U \longrightarrow$
*analz* $(H \cup A \cup$ *spies evsR5*$) = H \cup$ *analz* $(A \cup$ *spies evsR5*$)$ **and**
*B*: $($*evsR5*, *S*, *A*, *U*$) \in$ *protocol* **and**
*C*: *IntAgrK* $($*S* $($*Card n*, *n*, *run*$)) =$ *Some d* **and**
*D*: *ExtAgrK* $($*S* $($*Card n*, *n*, *run*$)) =$ *Some e*

**let**
*?B* = *H* ∪ *A* ∪ *spies evsR5* **and**
*?C* = *A* ∪ *spies evsR5*
**show** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) (*H* ∪ *A* ∪ *spies evsR5*)) =
*H* ∪ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) (*A* ∪ *spies evsR5*))
(**is** - ⟶ *?T*)
**proof** (*rule impI*, *cases Key* (*invK* (*sesK* (*d* ∗ *e*))) ∈ *analz ?C*)
**case** *True*
**assume** *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**with** *A* **have** *E*: *analz ?B* = *H* ∪ *analz ?C* **..**
**have** *Key* (*invK* (*sesK* (*d* ∗ *e*))) ∈ *analz ?B*
**by** (*rule subsetD* [*OF - True*], *rule analz-mono*, *blast*)
**hence** *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) *?B*) =
*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) (*insert* (*Number 0*) (*analz ?B*))
**by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**moreover have** *H* ∪ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) *?C*) =
*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) (*insert* (*Number 0*) (*H* ∪ *analz ?C*))
**using** *True* **by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**ultimately show** *?T*
**using** *E* **by** *simp*
**next**
**case** *False*
**moreover assume** *E*: *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U*
**with** *A* **have** *F*: *analz ?B* = *H* ∪ *analz ?C* **..**
**ultimately have** *Key* (*invK* (*sesK* (*d* ∗ *e*))) ∉ *analz ?B*
**proof** (*simp add*: *invK-sesK*, *insert pr-sesk-card* [*OF B C D*])
**qed** (*rule notI*, *drule subsetD* [*OF E*], *simp*)
**hence** *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) *?B*) =
*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) (*analz ?B*)
**by** (*simp add*: *analz-crypt-out*)
**moreover have** *H* ∪ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) *?C*) =
*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) (*H* ∪ *analz ?C*)
**using** *False* **by** (*simp add*: *analz-crypt-out*)
**ultimately show** *?T*
**using** *F* **by** *simp*
**qed**
**next**
**fix** *evsFR5 S A U m n run c f H*
**assume**
*A*: ⋀*H*. *H* ⊆ *range Key* ∪ *range Pri-AgrK* − *U* ⟶
*analz* (*H* ∪ *A* ∪ *spies evsFR5*) = *H* ∪ *analz* (*A* ∪ *spies evsFR5*) **and**
*B*: (*evsFR5*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* **and**
*D*: *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f*
**let**
*?B* = *H* ∪ *A* ∪ *spies evsFR5* **and**
*?C* = *A* ∪ *spies evsFR5*

**show** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U \longrightarrow$
$analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ (H \cup A \cup spies\ evsFR5)) =$
$H \cup analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ (A \cup spies\ evsFR5))$
(**is** - $\longrightarrow$ *?T*)
**proof** (*rule impI*, *cases Key* $(invK\ (sesK\ (c * f))) \in analz\ ?C$)
**case** *True*
**assume** $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**with** *A* **have** *E*: $analz\ ?B = H \cup analz\ ?C$ **..**
**have** $Key\ (invK\ (sesK\ (c * f))) \in analz\ ?B$
**by** (*rule subsetD* [*OF - True*], *rule analz-mono*, *blast*)
**hence** $analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ ?B) =$
$insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ (insert\ (Number\ 0)\ (analz\ ?B))$
**by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**moreover have** $H \cup analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ ?C) =$
$insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ (insert\ (Number\ 0)\ (H \cup analz$
*?C*))
**using** *True* **by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**ultimately show** *?T*
**using** *E* **by** *simp*
**next**
**case** *False*
**moreover assume** *E*: $H \subseteq range\ Key \cup range\ Pri\text{-}AgrK - U$
**with** *A* **have** *F*: $analz\ ?B = H \cup analz\ ?C$ **..**
**ultimately have** $Key\ (invK\ (sesK\ (c * f))) \notin analz\ ?B$
**proof** (*simp add*: *invK-sesK*, *insert pr-sesk-user-1* [*OF B C D*,
*THEN pr-sesk-user-2* [*OF B*]])
**qed** (*rule notI*, *drule subsetD* [*OF E*], *simp*)
**hence** $analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ ?B) =$
$insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ (analz\ ?B)$
**by** (*simp add*: *analz-crypt-out*)
**moreover have** $H \cup analz\ (insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ ?C) =$
$insert\ (Crypt\ (sesK\ (c * f))\ (Number\ 0))\ (H \cup analz\ ?C)$
**using** *False* **by** (*simp add*: *analz-crypt-out*)
**ultimately show** *?T*
**using** *F* **by** *simp*
**qed**
**qed**

**lemma** *pr-key-unused*:
$(evs, S, A, U) \in protocol \Longrightarrow$
$Key\ K \notin U \Longrightarrow$
$analz\ (insert\ (Key\ K)\ (A \cup spies\ evs)) =$
$insert\ (Key\ K)\ (analz\ (A \cup spies\ evs))$
**by** (*simp only*: *insert-def Un-assoc* [*symmetric*], *rule pr-key-set-unused*, *simp-all*)

**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-AgrK x*) (*analz* (*A* ∪ *spies evs*))
**by** (*simp only*: *insert-def Un-assoc* [*symmetric*], *rule pr-key-set-unused*, *simp-all*)

**lemma** *pr-pri-agrk-analz-intro* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*Pri-AgrK x* ∈ *analz* (*A* ∪ *spies evs*) ⟶
*Pri-AgrK x* ∈ *A*
**proof** (*erule protocol.induct*, *subst analz-simp*, *simp*, *blast*,
*simp-all add*: *analz-simp-insert-2 pr-key-unused pr-pri-agrk-unused*,
*rule conjI*, *rule-tac* [*1−2*] *impI*, *rule-tac* [!] *impI*)
**fix** *evsR1 S A U n s*
**assume**
*A*: *Pri-AgrK x* ∈ *analz* (*A* ∪ *spies evsR1*) ⟶ *Pri-AgrK x* ∈ *A*
(**is** - ∈ *analz ?A* ⟶ -) **and**
*B*: (*evsR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *n* ∈ *bad* **and**
*D*: *Pri-AgrK x* ∈ *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*))
(*insert* (*Pri-AgrK s*) (*A* ∪ *spies evsR1*))) **and**
*E*: *Pri-AgrK s* ∉ *U*
**have** *Key* (*symK n*) ∈ *analz ?A*
**using** *B* **and** *C* **by** (*simp add*: *pr-symk-analz*)
**hence** *Key* (*invK* (*symK n*)) ∈ *analz ?A*
**by** (*simp add*: *invK-symK*)
**hence** *Key* (*invK* (*symK n*)) ∈ *analz* (*insert* (*Pri-AgrK s*) *?A*)
**using** *B* **and** *E* **by** (*simp add*: *pr-pri-agrk-unused*)
**hence** *Pri-AgrK x* ∈ *analz* (*insert* (*Pri-AgrK s*) *?A*)
**using** *D* **by** (*simp add*: *analz-crypt-in*)
**hence** *x* = *s* ∨ *Pri-AgrK x* ∈ *analz ?A*
**using** *B* **and** *E* **by** (*simp add*: *pr-pri-agrk-unused*)
**thus** *x* = *s* ∨ *Pri-AgrK x* ∈ *A*
**using** *A* **by** *blast*
**next**
**fix** *evsR1 S A U n s*
**assume**
*A*: *Pri-AgrK x* ∈ *analz* (*A* ∪ *spies evsR1*) ⟶ *Pri-AgrK x* ∈ *A*
(**is** - ∈ *analz ?A* ⟶ -) **and**
*B*: (*evsR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *n* ∉ *bad* **and**
*D*: *Pri-AgrK x* ∈ *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*))
(*A* ∪ *spies evsR1*))
**have** *Key* (*symK n*) ∉ *analz ?A*
**using** *B* **and** *C* **by** (*simp add*: *pr-symk-analz*)
**hence** *Key* (*invK* (*symK n*)) ∉ *analz ?A*
**by** (*simp add*: *invK-symK*)
**hence** *Pri-AgrK x* ∈ *analz ?A*
**using** *D* **by** (*simp add*: *analz-crypt-out*)
**with** *A* **show** *Pri-AgrK x* ∈ *A* **..**
**next**
**fix** *evsFR1 A m s*

**assume**
*A*: *Pri-AgrK x* ∈ *analz* (*A* ∪ *spies evsFR1*) ⟶ *Pri-AgrK x* ∈ *A*
(**is** - ∈ *analz ?A* ⟶ -) **and**
*B*: *Crypt* (*symK m*) (*Pri-AgrK s*) ∈ *synth* (*analz* (*A* ∪ *spies evsFR1*)) **and**
*C*: *Pri-AgrK x* ∈ *analz* (*insert* (*Crypt* (*symK m*) (*Pri-AgrK s*))
(*A* ∪ *spies evsFR1*))
**show** *Pri-AgrK x* ∈ *A*
**proof** (*cases Key* (*invK* (*symK m*)) ∈ *analz ?A*)
**case** *True*
**have** *Crypt* (*symK m*) (*Pri-AgrK s*) ∈ *analz ?A* ∨
*Pri-AgrK s* ∈ *synth* (*analz ?A*) ∧ *Key* (*symK m*) ∈ *analz ?A*
**using** *B* **by** (*rule synth-crypt*)
**moreover {**
**assume** *Crypt* (*symK m*) (*Pri-AgrK s*) ∈ *analz ?A*
**hence** *Pri-AgrK s* ∈ *analz ?A*
**using** *True* **by** (*rule analz.Decrypt*)
**}**
**moreover {**
**assume** *Pri-AgrK s* ∈ *synth* (*analz ?A*) ∧ *Key* (*symK m*) ∈ *analz ?A*
**hence** *Pri-AgrK s* ∈ *synth* (*analz ?A*) **..**
**hence** *Pri-AgrK s* ∈ *analz ?A*
**by** (*rule synth-simp-intro*, *simp*)
**}**
**ultimately have** *Pri-AgrK s* ∈ *analz ?A* **..**
**moreover have** *Pri-AgrK x* ∈ *analz* (*insert* (*Pri-AgrK s*) *?A*)
**using** *C* **and** *True* **by** (*simp add*: *analz-crypt-in*)
**ultimately have** *Pri-AgrK x* ∈ *analz ?A*
**by** (*simp add*: *analz-simp-insert-1*)
**with** *A* **show** *?thesis* **..**
**next**
**case** *False*
**hence** *Pri-AgrK x* ∈ *analz ?A*
**using** *C* **by** (*simp add*: *analz-crypt-out*)
**with** *A* **show** *?thesis* **..**
**qed**
**next**
**fix** *evsC4 A c f*
**assume**
*A*: *Pri-AgrK x* ∈ *analz* (*A* ∪ *spies evsC4*) ⟶ *Pri-AgrK x* ∈ *A*
(**is** - ∈ *analz ?A* ⟶ -) **and**
*B*: *Pri-AgrK x* ∈ *analz* (*insert* (*Crypt* (*sesK* (*c* ∗ *f*)) (*pubAK f*))
(*A* ∪ *spies evsC4*))
**show** *Pri-AgrK x* ∈ *A*
**proof** (*cases Key* (*invK* (*sesK* (*c* ∗ *f*))) ∈ *analz ?A*)
**case** *True*
**hence** *Pri-AgrK x* ∈ *analz ?A*
**using** *B* **by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**with** *A* **show** *?thesis* **..**
**next**

**case** *False*
**hence** *Pri-AgrK x* ∈ *analz ?A*
**using** *B* **by** (*simp add*: *analz-crypt-out*)
**with** *A* **show** *?thesis* **..**
**qed**
**next**
**fix** *evsFC4 A s a b d e*
**assume**
*A*: *Pri-AgrK x* ∈ *analz* (*A* ∪ *spies evsFC4*) ⟶ *Pri-AgrK x* ∈ *A*
(**is** - ∈ *analz ?A* ⟶ -) **and**
*B*: *Pri-AgrK x* ∈ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK* (*d* ∗ (*s* + *a* ∗ *b*))))
(*A* ∪ *spies evsFC4*))
**show** *Pri-AgrK x* ∈ *A*
**proof** (*cases Key* (*invK* (*sesK* (*d* ∗ *e*))) ∈ *analz ?A*)
**case** *True*
**hence** *Pri-AgrK x* ∈ *analz ?A*
**using** *B* **by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**with** *A* **show** *?thesis* **..**
**next**
**case** *False*
**hence** *Pri-AgrK x* ∈ *analz ?A*
**using** *B* **by** (*simp add*: *analz-crypt-out*)
**with** *A* **show** *?thesis* **..**
**qed**
**next**
**fix** *evsR4 S A U n run b d e*
**let**
*?H* = *Hash* (*pubAK* (*priAK n*)) **and**
*?M* = {|*pubAK* (*priAK n*), *Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK n*)))|} **and**
*?M′* = {|*pubAK e*, *Auth-Data* (*priAK n*) *b*, *pubAK* (*priAK n*),
*Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK n*)))|}
**assume**
*A*: *Pri-AgrK x* ∈ *analz* (*A* ∪ *spies evsR4*) ⟶ *Pri-AgrK x* ∈ *A*
(**is** - ∈ *analz ?A* ⟶ -) **and**
*B*: (*evsR4*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b* **and**
*D*: *Pri-AgrK x* ∈ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) *?M′*)
(*A* ∪ *spies evsR4*))
**show** *Pri-AgrK x* ∈ *A*
**proof** (*cases Key* (*invK* (*sesK* (*d* ∗ *e*))) ∈ *analz ?A*)
**case** *True*
**have** *Key* (*pubSK CA*) ∈ *analz ?A*
**using** *B* **by** (*rule pr-valid-key-analz*)
**hence** *E*: *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?A*) =
{*Crypt* (*priSK CA*) *?H*, *?H*} ∪ *analz ?A*
**by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**have** *Pri-AgrK* (*priAK n*) ∉ *analz ?A*
**using** *B* **by** (*rule pr-auth-key-analz*)

```
    hence F: Pri-AgrK (priAK n) ∉ analz (insert (Crypt (priSK CA) ?H) ?A)
     using E by simp
    have Pri-AgrK b ∉ analz ?A
     using B and C by (rule pr-int-mapk-analz)
    hence G: Pri-AgrK b ∉ analz (insert (Crypt (priSK CA) ?H) ?A)
     using E by simp
    have Pri-AgrK x ∈ analz (insert ?M' ?A)
     using D and True by (simp add: analz-crypt-in)
    hence Pri-AgrK x ∈ analz (insert (Auth-Data (priAK n) b) (insert ?M ?A))
     by (simp add: analz-mpair analz-simp-insert-2)
    hence Pri-AgrK x ∈ analz ?A
    proof (subst (asm) insert-commute, simp add: analz-mpair analz-simp-insert-2
     del: Un-insert-right, subst (asm) insert-commute,
     subst (asm) analz-auth-data-out [OF F G])
    qed (simp add: E)
    with A show ?thesis ..
  next
    case False
    hence Pri-AgrK x ∈ analz ?A
     using D by (simp add: analz-crypt-out)
    with A show ?thesis ..
  qed
next
  fix evsFR4 S A U m n run s a b c f g
  let
    ?H = Hash (pubAK g) and
    ?M = {|pubAK g, Crypt (priSK CA) (Hash (pubAK g))|} and
    ?M' = {|pubAK (c * (s + a * b)), Auth-Data g b, pubAK g,
      Crypt (priSK CA) (Hash (pubAK g))|}
  assume
    A: Pri-AgrK x ∈ analz (A ∪ spies evsFR4) ⟶ Pri-AgrK x ∈ A
      (is - ∈ analz ?A ⟶ -) and
    B: (evsFR4, S, A, U) ∈ protocol and
    C: Crypt (sesK (c * f)) ?M' ∈ synth (analz (A ∪ spies evsFR4)) and
    D: Pri-AgrK x ∈ analz (insert (Crypt (sesK (c * f)) ?M')
      (A ∪ spies evsFR4))
  have E:
   Key (invK (sesK (c * f))) ∈ analz ?A ⟶
     Pri-AgrK b ∈ analz ?A ∨ Pri-AgrK g ∈ analz ?A ⟶
    Pri-AgrK b ∈ analz ?A ∧ Pri-AgrK g ∈ analz ?A
    (is ?P ⟶ ?Q ⟶ ?R)
  proof (rule impI)+
    assume ?P and ?Q
    have Crypt (sesK (c * f)) ?M' ∈ analz ?A ∨
      ?M' ∈ synth (analz ?A) ∧ Key (sesK (c * f)) ∈ analz ?A
     using C by (rule synth-crypt)
    moreover {
      assume Crypt (sesK (c * f)) ?M' ∈ analz ?A
      hence ?M' ∈ analz ?A
```

```
  using ‹?P› by (rule analz.Decrypt)
 hence {|Auth-Data g b, pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
   ∈ analz ?A
  by (rule analz.Snd)
 hence F: Auth-Data g b ∈ analz ?A
  by (rule analz.Fst)
 have ?R
 proof (rule disjE [OF ‹?Q›])
   assume Pri-AgrK b ∈ analz ?A
   moreover from this have Pri-AgrK g ∈ analz ?A
    by (rule analz.Auth-Fst [OF F])
   ultimately show ?R ..
 next
   assume Pri-AgrK g ∈ analz ?A
   moreover from this have Pri-AgrK b ∈ analz ?A
    by (rule analz.Auth-Snd [OF F])
   ultimately show ?R
    by simp
 qed
}
moreover {
 assume ?M' ∈ synth (analz ?A) ∧ Key (sesK (c * f)) ∈ analz ?A
 hence ?M' ∈ synth (analz ?A) ..
 hence {|Auth-Data g b, pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
   ∈ synth (analz ?A)
  by (rule synth-analz-snd)
 hence Auth-Data g b ∈ synth (analz ?A)
  by (rule synth-analz-fst)
 hence Auth-Data g b ∈ analz ?A ∨
   Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b ∈ analz ?A
  by (rule synth-auth-data)
 moreover {
   assume F: Auth-Data g b ∈ analz ?A
   have ?R
   proof (rule disjE [OF ‹?Q›])
     assume Pri-AgrK b ∈ analz ?A
     moreover from this have Pri-AgrK g ∈ analz ?A
      by (rule analz.Auth-Fst [OF F])
     ultimately show ?R ..
   next
     assume Pri-AgrK g ∈ analz ?A
     moreover from this have Pri-AgrK b ∈ analz ?A
      by (rule analz.Auth-Snd [OF F])
     ultimately show ?R
      by simp
   qed
 }
 moreover {
   assume Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b ∈ analz ?A
```

```
    hence ?R
     by simp
   }
   ultimately have ?R ..
  }
  ultimately show ?R ..
qed
show Pri-AgrK x ∈ A
proof (cases Key (invK (sesK (c * f))) ∈ analz ?A)
  case True
  have Key (pubSK CA) ∈ analz ?A
   using B by (rule pr-valid-key-analz)
  hence F: analz (insert (Crypt (priSK CA) ?H) ?A) =
    {Crypt (priSK CA) ?H, ?H} ∪ analz ?A
   by (simp add: analz-crypt-in analz-simp-insert-2)
  show ?thesis
  proof (cases Pri-AgrK g ∈ analz ?A ∨ Pri-AgrK b ∈ analz ?A, simp-all)
   assume G: Pri-AgrK g ∈ analz ?A ∨ Pri-AgrK b ∈ analz ?A
   hence H: Pri-AgrK g ∈ analz (insert (Crypt (priSK CA) ?H) ?A) ∨
     Pri-AgrK b ∈ analz (insert (Crypt (priSK CA) ?H) ?A)
    using F by simp
   have I: Pri-AgrK b ∈ analz ?A ∧ Pri-AgrK g ∈ analz ?A
    using E and G and True by blast
   hence Pri-AgrK g ∈ analz (insert (Crypt (priSK CA) ?H) ?A)
    using F by simp
   hence J: Pri-AgrK g ∈ analz (insert (Pri-AgrK b)
     (insert (Crypt (priSK CA) ?H) ?A))
    by (rule rev-subsetD, rule-tac analz-mono, blast)
   have K: Pri-AgrK b ∈ analz (insert (Crypt (priSK CA) ?H) ?A)
    using F and I by simp
   have Pri-AgrK x ∈ analz (insert ?M' ?A)
    using D and True by (simp add: analz-crypt-in)
   hence Pri-AgrK x ∈ analz (insert (Auth-Data g b) (insert ?M ?A))
    by (simp add: analz-mpair analz-simp-insert-2)
   hence Pri-AgrK x ∈ analz ?A
  proof (subst (asm) insert-commute, simp add: analz-mpair analz-simp-insert-2
    del: Un-insert-right, subst (asm) insert-commute,
    subst (asm) analz-auth-data-in [OF H], simp del: Un-insert-right)
     assume Pri-AgrK x ∈ analz (insert (Pri-AgrK g) (insert (Pri-AgrK b)
       (insert (Crypt (priSK CA) ?H) ?A)))
     thus ?thesis
     proof (subst (asm) analz-simp-insert-1 [OF J],
      subst (asm) analz-simp-insert-1 [OF K])
     qed (simp add: F)
   qed
   with A show ?thesis ..
  next
   assume G: Pri-AgrK g ∉ analz ?A ∧ Pri-AgrK b ∉ analz ?A
   hence H: Pri-AgrK g ∉ analz (insert (Crypt (priSK CA) ?H) ?A)
```

```
      using F by simp
     have I: Pri-AgrK b ∉ analz (insert (Crypt (priSK CA) ?H) ?A)
      using F and G by simp
     have Pri-AgrK x ∈ analz (insert ?M' ?A)
      using D and True by (simp add: analz-crypt-in)
     hence Pri-AgrK x ∈ analz (insert (Auth-Data g b) (insert ?M ?A))
      by (simp add: analz-mpair analz-simp-insert-2)
     hence Pri-AgrK x ∈ analz ?A
    proof (subst (asm) insert-commute, simp add: analz-mpair analz-simp-insert-2
      del: Un-insert-right, subst (asm) insert-commute,
      subst (asm) analz-auth-data-out [OF H I])
     qed (simp add: F)
     with A show ?thesis ..
   qed
  next
   case False
   hence Pri-AgrK x ∈ analz ?A
    using D by (simp add: analz-crypt-out)
   with A show ?thesis ..
  qed
next
  fix evsC5 A m c f
  assume
   A: Pri-AgrK x ∈ analz (A ∪ spies evsC5) ⟶ Pri-AgrK x ∈ A
     (is - ∈ analz ?A ⟶ -) and
   B: Pri-AgrK x ∈ analz (insert (Crypt (sesK (c * f)) (Passwd m))
     (A ∪ spies evsC5))
  show Pri-AgrK x ∈ A
  proof (cases Key (invK (sesK (c * f))) ∈ analz ?A)
   case True
   hence Pri-AgrK x ∈ analz ?A
    using B by (simp add: analz-crypt-in analz-simp-insert-2)
   with A show ?thesis ..
  next
   case False
   hence Pri-AgrK x ∈ analz ?A
    using B by (simp add: analz-crypt-out)
   with A show ?thesis ..
  qed
next
  fix evsFC5 A n d e
  assume
   A: Pri-AgrK x ∈ analz (A ∪ spies evsFC5) ⟶ Pri-AgrK x ∈ A
     (is - ∈ analz ?A ⟶ -) and
   B: Pri-AgrK x ∈ analz (insert (Crypt (sesK (d * e)) (Passwd n))
     (A ∪ spies evsFC5))
  show Pri-AgrK x ∈ A
  proof (cases Key (invK (sesK (d * e))) ∈ analz ?A)
   case True
```

```
    hence Pri-AgrK x ∈ analz ?A
     using B by (simp add: analz-crypt-in analz-simp-insert-2)
    with A show ?thesis ..
  next
    case False
    hence Pri-AgrK x ∈ analz ?A
     using B by (simp add: analz-crypt-out)
    with A show ?thesis ..
  qed
next
  fix evsR5 A d e
  assume
    A: Pri-AgrK x ∈ analz (A ∪ spies evsR5) ⟶ Pri-AgrK x ∈ A
      (is - ∈ analz ?A ⟶ -) and
    B: Pri-AgrK x ∈ analz (insert (Crypt (sesK (d * e)) (Number 0))
      (A ∪ spies evsR5))
  show Pri-AgrK x ∈ A
  proof (cases Key (invK (sesK (d * e))) ∈ analz ?A)
    case True
    hence Pri-AgrK x ∈ analz ?A
     using B by (simp add: analz-crypt-in analz-simp-insert-2)
    with A show ?thesis ..
  next
    case False
    hence Pri-AgrK x ∈ analz ?A
     using B by (simp add: analz-crypt-out)
    with A show ?thesis ..
  qed
next
  fix evsFR5 A c f
  assume
    A: Pri-AgrK x ∈ analz (A ∪ spies evsFR5) ⟶ Pri-AgrK x ∈ A
      (is - ∈ analz ?A ⟶ -) and
    B: Pri-AgrK x ∈ analz (insert (Crypt (sesK (c * f)) (Number 0))
      (A ∪ spies evsFR5))
  show Pri-AgrK x ∈ A
  proof (cases Key (invK (sesK (c * f))) ∈ analz ?A)
    case True
    hence Pri-AgrK x ∈ analz ?A
     using B by (simp add: analz-crypt-in analz-simp-insert-2)
    with A show ?thesis ..
  next
    case False
    hence Pri-AgrK x ∈ analz ?A
     using B by (simp add: analz-crypt-out)
    with A show ?thesis ..
  qed
qed
```

**lemma** *pr-pri-agrk-analz*:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
(*Pri-AgrK x* ∈ *analz* (*A* ∪ *spies evs*)) = (*Pri-AgrK x* ∈ *A*)
**proof** (*rule iffI*, *erule pr-pri-agrk-analz-intro*, *assumption*)
**qed** (*rule subsetD* [*OF analz-subset*], *simp*)

**lemma** *pr-ext-agrk-user-1* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*User m* ≠ *Spy* ⟶
*Says n run* 4 (*User m*) (*Card n*) (*Crypt* (*sesK K*) (*pubAK e*)) ∈ *set evs* ⟶
*ExtAgrK* (*S* (*User m*, *n*, *run*)) ≠ *None*
**by** (*erule protocol.induct*, *simp-all*, (*rule-tac* [!] *impI*)+, *simp-all*)

**lemma** *pr-ext-agrk-user-2* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*User m* ≠ *Spy* ⟶
*Says n run* 4 *X* (*User m*) (*Crypt* (*sesK K*)
{|*pubAK e*, *Auth-Data x y*, *pubAK g*, *Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|})
∈ *set evs* ⟶
*ExtAgrK* (*S* (*User m*, *n*, *run*)) ≠ *None*
**using** [[*simproc del*: *defined-all*]] **proof** (*erule protocol.induct*, *simp-all*, (*rule-tac* [!] *impI*)+, *simp-all*,
(*erule conjE*)+)
**fix** *evs S A U n run s a b d e X*
**assume** (*evs*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *0* < *m*
**hence** *User m* ≠ *Spy*
**by** *simp*
**moreover assume** *A*: *User m* = *X* **and**
*Says n run* 4 *X* (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*))
(*pubAK* (*d* ∗ (*s* + *a* ∗ *b*)))) ∈ *set evs*
**hence** *Says n run* 4 (*User m*) (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*))
(*pubAK* (*d* ∗ (*s* + *a* ∗ *b*)))) ∈ *set evs*
**by** *simp*
**ultimately have** *ExtAgrK* (*S* (*User m*, *n*, *run*)) ≠ *None*
**by** (*rule pr-ext-agrk-user-1*)
**thus** ∃ *e*. *ExtAgrK* (*S* (*X*, *n*, *run*)) = *Some e*
**using** *A* **by** *simp*
**qed**

**lemma** *pr-ext-agrk-user-3* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*User m* ≠ *Spy* ⟶
*ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some e* ⟶
*Says n run* 4 (*User m*) (*Card n*) (*Crypt* (*sesK K*) (*pubAK e*′)) ∈ *set evs* ⟶
*e*′ = *e*
**proof** (*erule protocol.induct*, *simp-all*, (*rule conjI*, (*rule-tac* [!] *impI*)+)+,
(*erule conjE*)+, *simp-all*)
**assume** *agrK e*′ = *agrK e*

**with** *agrK-inj* **show** $e' = e$
**by** (*rule injD*)
**next**
**fix** *evsC4 S A U n run* **and** $m$ :: *nat*
**assume** (*evsC4*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** $0 < m$
**hence** *User m* ≠ *Spy*
**by** *simp*
**moreover assume**
*Says n run 4* (*User m*) (*Card n*) (*Crypt* (*sesK K*) (*pubAK e'*)) ∈ *set evsC4*
**ultimately have** *ExtAgrK* (*S* (*User m*, *n*, *run*)) ≠ *None*
**by** (*rule pr-ext-agrk-user-1*)
**moreover assume** *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *None*
**ultimately show** $e' = e$
**by** *contradiction*
**qed**

**lemma** *pr-ext-agrk-user-4* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f* ⟶
(∃ *X*. *Says n run 3 X* (*User m*) (*pubAK f*) ∈ *set evs*)
**proof** (*erule protocol.induct*, *simp-all*, *rule-tac* [!] *impI*, *rule-tac* [*1−2*] *impI*,
*rule-tac* [*5*] *impI*, *simp-all*)
**qed** *blast*+

**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*)
**apply** (*erule protocol.induct*, *simp-all add*: *pr-pri-agrk-analz*)
**apply** (*rule conjI*)
**apply** (*rule-tac* [*1−2*] *impI*)+
**apply** (*rule-tac* [*3*] *conjI*, (*rule-tac* [*3*] *impI*)+)+
**apply** (*rule-tac* [*4*] *impI*)+
**apply** ((*rule-tac* [*5*] *impI*)+, (*rule-tac* [*5*] *conjI*)*?*)+
**apply** (*rule-tac* [*6*] *impI*)+
**apply** ((*rule-tac* [*7*] *impI*)+, (*rule-tac* [*7*] *conjI*)*?*)+
**apply** (*rule-tac* [*8*] *impI*)+
**apply** ((*rule-tac* [*9*] *impI*)+, (*rule-tac* [*9*] *conjI*)*?*)+

```
            apply (rule-tac [10] impI)+
            apply (rule-tac [11] impI)+
            apply (rule-tac [12] conjI, (rule-tac [12] impI)+)+
             apply (rule-tac [13] impI)+
             apply (rule-tac [14] conjI, (rule-tac [14] impI)+)+
               apply (erule-tac [14] conjE)+
               apply (rule-tac [15] impI)+
               apply ((rule-tac [16] impI)+, (rule-tac [16] conjI)?)+
                apply (erule-tac [16] conjE)+
                apply (rule-tac [17-18] impI)
proof -
  fix evsR1 S A U s n' run'
  assume
   ?H evsR1 ⟶ ?P S n run ∨ ?Q S A n run and
   ?H evsR1
  hence A: ?P S n run ∨ ?Q S A n run ..
  assume B: NonceS (S (Card n', n', run')) = None
  let ?S = S((Card n', n', run') := S (Card n', n', run')
    (|NonceS := Some s|))
  show ?P ?S n run ∨
    (∃ b. (b = s ∨ Pri-AgrK b ∈ A) ∧ ExtMapK (?S (User m, n, run)) = Some b)
  proof (rule disjE [OF A], rule disjI1, simp add: fun-upd-apply, rule impI, simp
add: B)
  qed (rule disjI2, simp add: fun-upd-apply, blast)
next
  fix evsR1 S A U s n' run'
  assume
   ?H evsR1 ⟶ ?P S n run ∨ ?Q S A n run and
   ?H evsR1
  hence A: ?P S n run ∨ ?Q S A n run ..
  assume B: NonceS (S (Card n', n', run')) = None
  let ?S = S((Card n', n', run') := S (Card n', n', run')
    (|NonceS := Some s|))
  show ?P ?S n run ∨ ?Q ?S A n run
  proof (rule disjE [OF A], rule disjI1, simp add: fun-upd-apply, rule impI, simp
add: B)
  qed (rule disjI2, simp add: fun-upd-apply)
next
  fix evsC2 S A U s a n' run'
  assume
   ?H evsC2 ⟶ ?P S n run ∨ ?Q S A n run and
   ?H evsC2
  hence A: ?P S n run ∨ ?Q S A n run ..
  let ?S = S((Spy, n', run') := S (Spy, n', run')
    (|NonceS := Some s, IntMapK := Some a|))
  show ?P ?S n run ∨
    (∃ b. (b = a ∨ Pri-AgrK b ∈ A) ∧ ExtMapK (?S (User m, n, run)) = Some b)
   by (rule disjE [OF A], simp-all add: fun-upd-apply, blast)
next
```

**fix** *evsC2 S A U s a m′ n′ run′*
**assume**
*?H evsC2* $\longrightarrow$ *?P S n run* $\vee$ *?Q S A n run* **and**
*?H evsC2*
**hence** *A*: *?P S n run* $\vee$ *?Q S A n run* **..**
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* $\vee$ *?Q ?S A n run*
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a n′ run′*
**assume**
*?H evsC2* $\longrightarrow$ *?P S n run* $\vee$ *?Q S A n run* **and**
*?H evsC2*
**hence** *A*: *?P S n run* $\vee$ *?Q S A n run* **..**
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* $\vee$
($\exists$ *b*. (*b* = *a* $\vee$ *Pri-AgrK b* $\in$ *A*) $\wedge$ *ExtMapK* (*?S* (*User m*, *n*, *run*)) = *Some b*)
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a m′ n′ run′*
**assume**
*?H evsC2* $\longrightarrow$ *?P S n run* $\vee$ *?Q S A n run* **and**
*?H evsC2*
**hence** *A*: *?P S n run* $\vee$ *?Q S A n run* **..**
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* $\vee$ *?Q ?S A n run*
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a n′ run′*
**assume**
*?H evsC2* $\longrightarrow$ *?P S n run* $\vee$ *?Q S A n run* **and**
*?H evsC2*
**hence** *A*: *?P S n run* $\vee$ *?Q S A n run* **..**
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* $\vee$
($\exists$ *b*. (*b* = *a* $\vee$ *Pri-AgrK b* $\in$ *A*) $\wedge$ *ExtMapK* (*?S* (*User m*, *n*, *run*)) = *Some b*)
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a m′ n′ run′*
**assume**
*?H evsC2* $\longrightarrow$ *?P S n run* $\vee$ *?Q S A n run* **and**
*?H evsC2*
**hence** *A*: *?P S n run* $\vee$ *?Q S A n run* **..**
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))

**show** *?P ?S n run* ∨ *?Q ?S A n run*
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a n′ run′*
**assume**
*?H evsC2* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H evsC2*
**hence** *A*: *?P S n run* ∨ *?Q S A n run* **..**
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* ∨
(∃ *b*. (*b* = *a* ∨ *Pri-AgrK b* ∈ *A*) ∧ *ExtMapK* (*?S* (*User m*, *n*, *run*)) = *Some b*)
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a m′ n′ run′*
**assume**
*?H evsC2* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H evsC2*
**hence** *A*: *?P S n run* ∨ *?Q S A n run* **..**
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsR2 S A U a b n′ run′*
**assume**
*?H evsR2* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H evsR2*
**hence** *A*: *?P S n run* ∨ *?Q S A n run* **..**
**assume** *B*: *IntMapK* (*S* (*Card n′*, *n′*, *run′*)) = *None*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(|*IntMapK* := *Some b*, *ExtMapK* := *Some a*|))
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**proof** (*rule disjE* [*OF A*], *rule disjI1*, *simp add*: *fun-upd-apply*, *rule impI*, *simp add*: *B*)
**qed** (*rule disjI2*, *simp add*: *fun-upd-apply*)
**next**
**fix** *evsC3 S A U b c m′ n′ run′*
**assume**
*?H evsC3* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H evsC3*
**hence** *A*: *?P S n run* ∨ *?Q S A n run* **..**
**assume**
*ExtMapK* (*S* (*User m′*, *n′*, *run′*)) = *None* **and**
*m′* = *0*
**hence** *B*: *ExtMapK* (*S* (*Spy*, *n′*, *run′*)) = *None*
**by** *simp*
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*ExtMapK* := *Some b*, *IntAgrK* := *Some c*|))

**show** *?P ?S n run* ∨
(∃ *b*. (*b* = *c* ∨ *Pri-AgrK b* ∈ *A*) ∧ *ExtMapK* (*?S* (*User m*, *n*, *run*)) = *Some b*)
**proof** (*rule disjE* [*OF A*], *simp add*: *fun-upd-apply*)
**qed** (*rule disjI2*, *simp add*: *fun-upd-apply*, *rule conjI*, *rule impI*, *simp add*: *B*, *blast*)
**next**
**fix** *evsC3 S A U b c m′ n′ run′*
**assume**
*?H evsC3* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H evsC3*
**hence** *A*: *?P S n run* ∨ *?Q S A n run* **..**
**assume** *B*: *ExtMapK* (*S* (*User m′*, *n′*, *run′*)) = *None*
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*ExtMapK* := *Some b*, *IntAgrK* := *Some c*|))
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**proof** (*rule disjE* [*OF A*], *simp add*: *fun-upd-apply*)
**qed** (*rule disjI2*, *simp add*: *fun-upd-apply*, *rule impI*, *simp add*: *B*)
**next**
**fix** *evsR3 A U s a b c d n′ run′ X* **and** *S* :: *state*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(|*IntAgrK* := *Some d*, *ExtAgrK* := *Some* (*c* ∗ (*s* + *a* ∗ *b*))|))
**assume** *agrK f* = *agrK* (*d* ∗ (*s* + *a* ∗ *b*))
**with** *agrK-inj* **have** *f* = *d* ∗ (*s* + *a* ∗ *b*)
**by** (*rule injD*)
**moreover assume**
*NonceS* (*S* (*Card n′*, *n′*, *run′*)) = *Some s* **and**
*IntMapK* (*S* (*Card n′*, *n′*, *run′*)) = *Some b* **and**
*ExtMapK* (*S* (*Card n′*, *n′*, *run′*)) = *Some a* **and**
*d* ≠ *0* **and**
*s* + *a* ∗ *b* ≠ *0*
**ultimately show** *?P ?S n′ run′* ∨
(∃ *b*. *Pri-AgrK b* ∈ *A* ∧ *ExtMapK* (*?S* (*X*, *n′*, *run′*)) = *Some b*)
**by** (*simp add*: *fun-upd-apply*)
**next**
**fix** *evsR3 S A U s a b c d n′ run′*
**assume**
*?H evsR3* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H evsR3*
**hence** *A*: *?P S n run* ∨ *?Q S A n run* **..**
**assume** *B*: *IntAgrK* (*S* (*Card n′*, *n′*, *run′*)) = *None*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(|*IntAgrK* := *Some d*, *ExtAgrK* := *Some* (*c* ∗ (*s* + *a* ∗ *b*))|))
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**proof** (*rule disjE* [*OF A*], *rule disjI1*, *simp add*: *fun-upd-apply*, *rule impI*, *simp add*: *B*)
**qed** (*rule disjI2*, *simp add*: *fun-upd-apply*)
**next**
**fix** *evsR3 A U s s′ a b c d n′ run′ X* **and** *S* :: *state*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)

```
    (|IntAgrK := Some d, ExtAgrK := Some (c * (s' + a * b))|))
  assume agrK f = agrK (d * (s + a * b))
  with agrK-inj have f = d * (s + a * b)
   by (rule injD)
  moreover assume
   NonceS (S (Card n', n', run')) = Some s and
   IntMapK (S (Card n', n', run')) = Some b and
   ExtMapK (S (Card n', n', run')) = Some a and
   d ≠ 0 and
   s + a * b ≠ 0
  ultimately show ?P ?S n' run' ∨
    (∃ b. Pri-AgrK b ∈ A ∧ ExtMapK (?S (X, n', run')) = Some b)
   by (simp add: fun-upd-apply)
next
  fix evsR3 S A U s a b c d n' run'
  assume
   ?H evsR3 ⟶ ?P S n run ∨ ?Q S A n run and
   ?H evsR3
  hence A: ?P S n run ∨ ?Q S A n run ..
  assume B: IntAgrK (S (Card n', n', run')) = None
  let ?S = S((Card n', n', run') := S (Card n', n', run')
    (|IntAgrK := Some d, ExtAgrK := Some (c * (s + a * b))|))
  show ?P ?S n run ∨ ?Q ?S A n run
  proof (rule disjE [OF A], rule disjI1, simp add: fun-upd-apply, rule impI, simp
add: B)
  qed (rule disjI2, simp add: fun-upd-apply)
next
  fix evsC4 S A U f m' n' run'
  assume
   ?H evsC4 ⟶ ?P S n run ∨ ?Q S A n run and
   ?H evsC4
  hence A: ?P S n run ∨ ?Q S A n run ..
  let ?S = S((User m', n', run') := S (User m', n', run')
    (|ExtAgrK := Some f|))
  show ?P ?S n run ∨ ?Q ?S A n run
   by (rule disjE [OF A], simp-all add: fun-upd-apply, blast)
qed

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
by (erule protocol.induct, simp-all)

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 (erule protocol.induct, simp-all, rule-tac [3] conjI, (rule-tac [!] impI)+,
 rule-tac [2] impI, rule-tac [4] impI, rule-tac [!] notI, simp-all)
  fix evsR1 S A U s
  assume
   (evsR1, S, A, U) ∈ protocol and
   IntAgrK (S (User m, n, run)) = Some s
  hence Pri-AgrK s ∈ U
   by (rule pr-int-agrk-user-1)
  moreover assume Pri-AgrK s ∉ U
  ultimately show False
   by contradiction
next
  fix evsC2 S A U a
  assume
   (evsC2, S, A, U) ∈ protocol and
   IntAgrK (S (User m, n, run)) = Some a
  hence Pri-AgrK a ∈ U
   by (rule pr-int-agrk-user-1)
  moreover assume Pri-AgrK a ∉ U
  ultimately show False
   by contradiction
next
  fix evsC3 S A U
  assume (evsC3, S, A, U) ∈ protocol
  hence A ⊆ U
   by (rule pr-analz-used)
  moreover assume Pri-AgrK c ∈ A
  ultimately have Pri-AgrK c ∈ U ..
  moreover assume Pri-AgrK c ∉ U
  ultimately show False
   by contradiction
next
  fix evsC3 S A U c'
  assume
   (evsC3, S, A, U) ∈ protocol and
   IntAgrK (S (User m, n, run)) = Some c'
  hence Pri-AgrK c' ∈ U
   by (rule pr-int-agrk-user-1)
  moreover assume Pri-AgrK c' ∉ U
  ultimately show False
   by contradiction
qed

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** (*erule protocol.induct*, *simp-all*, *rule conjI*, (*rule-tac* [*1*−*2*] *impI*)+,
(*rule-tac* [*3*] *impI*)+, *simp-all*)
**fix** *evsC2 S A U n run m*
**assume** *A*: (*evsC2*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *NonceS* (*S* (*User m*, *n*, *run*)) = *None*
**ultimately have** *IntMapK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-1*)
**with** *A* **have** *ExtMapK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-2*)
**moreover assume** *ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b*
**ultimately show** *c* ≠ *0* ∧ *s* + *a* ∗ *b* ≠ *0*
**by** *simp*
**next**
**fix** *evsC2 S A U n run m*
**assume** *A*: (*evsC2*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *NonceS* (*S* (*User m*, *n*, *run*)) = *None*
**ultimately have** *IntMapK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-1*)
**with** *A* **have** *ExtMapK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-2*)
**moreover assume** *ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b*
**ultimately show** *c* ≠ *0* ∧ *a* ≠ *0* ∧ *b* ≠ *0*
**by** *simp*
**qed**

**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** (*erule protocol.induct*, *simp-all*, (*rule-tac* [!] *impI*)+, *simp-all*)
**fix** *evsR1 S A U n run*
**assume**
(*evsR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*NonceS* (*S* (*Card n*, *n*, *run*)) = *None*
**hence** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-1*)
**moreover assume** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b*
**ultimately show** *d* ≠ *0* ∧ *s* + *a* ∗ *b* ≠ *0*
**by** *simp*
**next**
**fix** *evsR2 S A U n run*
**assume** *A*: (*evsR2*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntMapK* (*S* (*Card n*, *n*, *run*)) = *None*

**hence** *ExtMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-2*)
**with** *A* **have** *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-3*)
**moreover assume** *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d*
**ultimately show** *d* ≠ *0* ∧ *s* + *a* ∗ *b* ≠ *0*
**by** *simp*
**qed**

**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*
**apply** (*erule protocol.induct*, *simp-all add*: *pr-pri-agrk-analz*)
**apply** (*rule conjI*)
**apply** (*rule-tac* [*1−2*] *impI*)+
**apply** (*rule-tac* [*3*] *impI*)+
**apply** (*rule-tac* [*4*] *conjI*, (*rule-tac* [*4*] *impI*)+)+
**apply** (*erule-tac* [*4*] *conjE*)+
**apply** (*rule-tac* [*5*] *impI*)
**apply** (*erule-tac* [*5*] *conjE*)+
**apply** (*rule-tac* [*6*] *impI*)+
**apply** (*erule-tac* [*6*] *conjE*)+
**apply** (*rule-tac* [*6*] *notI*)
**apply** ((*rule-tac* [*7*] *impI*)+, (*rule-tac* [*7*] *conjI*)*?*)+
**apply** (*erule-tac* [*7*] *conjE*)+
**apply** (*rule-tac* [*8*] *impI*)
**apply** (*erule-tac* [*8*] *conjE*)+
**apply** (*rule-tac* [*9*] *impI*)+
**apply** (*rule-tac* [*9*] *notI*)
**apply** (*rule-tac* [*10*] *impI*)+
**apply** (*rule-tac* [*11*] *impI*)+
**apply** (*rule-tac* [*11*] *notI*)
**proof** *simp-all*
**fix** *evsR1 S A U n run*
**assume**
(*evsR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*NonceS* (*S* (*Card n*, *n*, *run*)) = *None*
**hence** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-1*)
**moreover assume** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b*
**ultimately show** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
**by** *simp*
**next**

**fix** *evsR1 S A U n run*
**assume**
(*evsR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*NonceS* (*S* (*Card n*, *n*, *run*)) = *None*
**hence** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-1*)
**moreover assume** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b*
**ultimately show** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
**by** *simp*
**next**
**fix** *evsR2 S A U n run*
**assume** *A*: (*evsR2*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**hence** *ExtMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-2*)
**with** *A* **have** *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-3*)
**moreover assume** *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d*
**ultimately show** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
**by** *simp*
**next**
**fix** *evsR3 S A U s a′ b′ c′ d′*
**assume**
(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some* (*c* ∗ (*s* + *a* ∗ *b*))
**hence** *Key* (*sesK* (*d* ∗ (*c* ∗ (*s* + *a* ∗ *b*)))) ∈ *U*
**by** (*rule pr-sesk-card*)
**moreover have** *d* ∗ (*c* ∗ (*s* + *a* ∗ *b*)) = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*)
**by** *simp*
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** *simp*
**moreover assume** *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*)) = *sesK* (*c′* ∗ *d′* ∗ (*s* + *a′* ∗ *b′*))
**ultimately have** *Key* (*sesK* (*c′* ∗ *d′* ∗ (*s* + *a′* ∗ *b′*))) ∈ *U*
**by** *simp*
**moreover assume** *Key* (*sesK* (*c′* ∗ *d′* ∗ (*s* + *a′* ∗ *b′*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsR3 S A U s′ a′ b′ c′ d′*
**assume**
(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some* (*c* ∗ (*s* + *a* ∗ *b*))
**hence** *Key* (*sesK* (*d* ∗ (*c* ∗ (*s* + *a* ∗ *b*)))) ∈ *U*
**by** (*rule pr-sesk-card*)
**moreover have** *d* ∗ (*c* ∗ (*s* + *a* ∗ *b*)) = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*)
**by** *simp*
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*

**by** *simp*
**moreover assume** *sesK* $(c * d * (s + a * b)) = sesK\ (c' * d' * (s' + a' * b'))$
**ultimately have** *Key* $(sesK\ (c' * d' * (s' + a' * b'))) \in U$
**by** *simp*
**moreover assume** *Key* $(sesK\ (c' * d' * (s' + a' * b'))) \notin U$
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsR3 S A U s' c'*
**assume** $(evsR3,\ S,\ A,\ U) \in protocol$
**hence** $A \subseteq U$
**by** (*rule pr-analz-used*)
**moreover assume** *Key* $(sesK\ (c' * d * (s' + a * b))) \notin U$
**ultimately have** *Key* $(sesK\ (c' * d * (s' + a * b))) \notin A$
**by** (*rule contra-subsetD*)
**moreover assume** $c' * (s' + a * b) = c * (s + a * b)$
**hence** $c' * d * (s' + a * b) = c * d * (s + a * b)$
**by** *simp*
**ultimately show** *Key* $(sesK\ (c * d * (s + a * b))) \notin A$
**by** *simp*
**next**
**fix** *evsFR3 S A U s' a' b' c' d'*
**assume**
$(evsFR3,\ S,\ A,\ U) \in protocol$ **and**
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d$ **and**
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ (c * (s + a * b))$
**hence** *Key* $(sesK\ (d * (c * (s + a * b)))) \in U$
**by** (*rule pr-sesk-card*)
**moreover have** $d * (c * (s + a * b)) = c * d * (s + a * b)$
**by** *simp*
**ultimately have** *Key* $(sesK\ (c * d * (s + a * b))) \in U$
**by** *simp*
**moreover assume** $sesK\ (c * d * (s + a * b)) = sesK\ (c' * d' * (s' + a' * b'))$
**ultimately have** *Key* $(sesK\ (c' * d' * (s' + a' * b'))) \in U$
**by** *simp*
**moreover assume** *Key* $(sesK\ (c' * d' * (s' + a' * b'))) \notin U$
**ultimately show** *False*
**by** *contradiction*
**qed**

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

**lemma** *pr-sesk-user-3* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$\{\!|Key\ (sesK\ K),\ Agent\ (User\ m),\ Number\ n,\ Number\ run|\!\} \in U \longrightarrow$
$Key\ (sesK\ K) \in A \longrightarrow$
$(\exists\, d\ e.\ K = d * e \wedge$
$IntAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ d \wedge$
$ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ e) \vee$

```
  (∃ 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)
apply (erule protocol.induct, simp-all add: pr-pri-agrk-analz, blast)
     apply (rule conjI)
      apply (rule-tac [1−2] impI)+
      apply (rule-tac [3] conjI, (rule-tac [3] impI)+)+
        apply (rule-tac [4] impI)+
        apply ((rule-tac [5] impI)+, (rule-tac [5] conjI)?)+
         apply (rule-tac [6] impI)+
         apply ((rule-tac [7] impI)+, (rule-tac [7] conjI)?)+
           apply (rule-tac [8] impI)+
           apply ((rule-tac [9] impI)+, (rule-tac [9] conjI)?)+
            apply (rule-tac [10] impI)+
            apply (rule-tac [11] impI)+
            apply (rule-tac [12] conjI, (rule-tac [12] impI)+)+
             apply (rule-tac [13] impI)+
             apply (rule-tac [14] conjI, (rule-tac [14] impI)+)+
               apply (erule-tac [14] conjE)+
               apply ((rule-tac [15] impI)+, (rule-tac [15] conjI)?)+
                apply (rule-tac [16] impI)+
                apply ((rule-tac [17] impI)+, (rule-tac [17] conjI)?)+
                 apply (rule-tac [18−20] impI)+
proof −
  fix evsR1 S A U s n′ run′
  assume
   ?H1 U ⟶ ?H2 A ⟶ ?P S n run ∨ ?Q S A n run and
   ?H1 U and
   ?H2 A
  hence A: ?P S n run ∨ ?Q S A n run
   by simp
  let ?S = S((Card n′, n′, run′) := S (Card n′, n′, run′)
    (|NonceS := Some s|))
  show ?P ?S n run ∨
    (∃ b. (b = s ∨ Pri-AgrK b ∈ A) ∧ ExtMapK (?S (User m, n, run)) = Some b)
  proof (rule disjE [OF A], rule disjI1, simp add: fun-upd-apply, rule impI, simp)
  qed (rule disjI2, simp add: fun-upd-apply, blast)
next
  fix evsR1 S A U s n′ run′
  assume
   ?H1 U ⟶ ?H2 A ⟶ ?P S n run ∨ ?Q S A n run and
   ?H1 U and
   ?H2 A
  hence A: ?P S n run ∨ ?Q S A n run
   by simp
  let ?S = S((Card n′, n′, run′) := S (Card n′, n′, run′)
    (|NonceS := Some s|))
  show ?P ?S n run ∨ ?Q ?S A n run
  proof (rule disjE [OF A], rule disjI1, simp add: fun-upd-apply, rule impI, simp)
```

**qed** (*rule disjI2*, *simp add*: *fun-upd-apply*)
**next**
**fix** *evsC2 S A U s a n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* ∨
(∃ *b*. (*b* = *a* ∨ *Pri-AgrK b* ∈ *A*) ∧ *ExtMapK* (*?S* (*User m*, *n*, *run*)) = *Some b*)
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a m n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*User m*, *n′*, *run′*) := *S* (*User m*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* ∨
(∃ *b*. (*b* = *a* ∨ *Pri-AgrK b* ∈ *A*) ∧ *ExtMapK* (*?S* (*User m*, *n*, *run*)) = *Some b*)
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a m n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*User m*, *n′*, *run′*) := *S* (*User m*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))

**show** *?P ?S n run* ∨ *?Q ?S A n run*
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* ∨
(∃ *b*. (*b* = *a* ∨ *Pri-AgrK b* ∈ *A*) ∧ *ExtMapK* (*?S* (*User m*, *n*, *run*)) = *Some b*)
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a m n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*User m*, *n′*, *run′*) := *S* (*User m*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**show** *?P ?S n run* ∨
(∃ *b*. (*b* = *a* ∨ *Pri-AgrK b* ∈ *A*) ∧ *ExtMapK* (*?S* (*User m*, *n*, *run*)) = *Some b*)
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)
**next**
**fix** *evsC2 S A U s a m n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*User m*, *n′*, *run′*) := *S* (*User m*, *n′*, *run′*)

```
    (|NonceS := Some s, IntMapK := Some a|))
  show ?P ?S n run ∨ ?Q ?S A n run
   by (rule disjE [OF A], simp-all add: fun-upd-apply, blast)
next
  fix evsR2 S A U a b n' run'
  assume
   ?H1 U ⟶ ?H2 A ⟶ ?P S n run ∨ ?Q S A n run and
   ?H1 U and
   ?H2 A
  hence A: ?P S n run ∨ ?Q S A n run
   by simp
  let ?S = S((Card n', n', run') := S (Card n', n', run')
    (|IntMapK := Some b, ExtMapK := Some a|))
  show ?P ?S n run ∨ ?Q ?S A n run
  proof (rule disjE [OF A], rule disjI1, simp add: fun-upd-apply, rule impI, simp)
  qed (rule disjI2, simp add: fun-upd-apply)
next
  fix evsC3 S A U b c m' n' run'
  assume
   ?H1 U ⟶ ?H2 A ⟶ ?P S n run ∨ ?Q S A n run and
   ?H1 U and
   ?H2 A
  hence A: ?P S n run ∨ ?Q S A n run
   by simp
  assume
   ExtMapK (S (User m', n', run')) = None and
   m' = 0
  hence B: ExtMapK (S (Spy, n', run')) = None
   by simp
  let ?S = S((Spy, n', run') := S (Spy, n', run')
    (|ExtMapK := Some b, IntAgrK := Some c|))
  show ?P ?S n run ∨
    (∃ b. (b = c ∨ Pri-AgrK b ∈ A) ∧ ExtMapK (?S (User m, n, run)) = Some b)
  proof (rule disjE [OF A], rule disjI1, simp add: fun-upd-apply)
  qed (rule disjI2, simp add: fun-upd-apply, rule conjI, rule impI, simp add: B,
blast)
next
  fix evsC3 S A U b c m' n' run'
  assume
   ?H1 U ⟶ ?H2 A ⟶ ?P S n run ∨ ?Q S A n run and
   ?H1 U and
   ?H2 A
  hence A: ?P S n run ∨ ?Q S A n run
   by simp
  assume B: ExtMapK (S (User m', n', run')) = None
  let ?S = S((User m', n', run') := S (User m', n', run')
    (|ExtMapK := Some b, IntAgrK := Some c|))
  show ?P ?S n run ∨ ?Q ?S A n run
  proof (rule disjE [OF A], rule disjI1, simp add: fun-upd-apply)
```

**qed** (*rule disjI2*, *simp add*: *fun-upd-apply*, *rule impI*, *simp add*: *B*)
**next**
**fix** *evsR3 A s a b c d n′ run′ X* **and** *S* :: *state*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(⦇*IntAgrK* := *Some d*, *ExtAgrK* := *Some* (*c* ∗ (*s* + *a* ∗ *b*))⦈)
**assume** *sesK K* = *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))
**with** *sesK-inj* **have** *K* = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*)
**by** (*rule injD*)
**thus** *?P ?S n′ run′* ∨
(∃ *b*. *Pri-AgrK b* ∈ *A* ∧ *ExtMapK* (*?S* (*X*, *n′*, *run′*)) = *Some b*)
**by** (*simp add*: *fun-upd-apply*)
**next**
**fix** *evsR3 A U s a b c d n′ run′* **and** *S* :: *state*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(⦇*IntAgrK* := *Some d*, *ExtAgrK* := *Some* (*c* ∗ (*s* + *a* ∗ *b*))⦈)
**assume** *sesK K* = *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))
**with** *sesK-inj* **have** *K* = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*)
**by** (*rule injD*)
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately have** *Key* (*sesK K*) ∉ *U*
**by** *simp*
**moreover assume**
(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
{|*Key* (*sesK K*), *Agent* (*User m*), *Number n*, *Number run*|} ∈ *U*
**hence** *Key* (*sesK K*) ∈ *U*
**by** (*rule pr-sesk-user-2*)
**ultimately show** *?P ?S n run* ∨ *?Q ?S A n run*
**by** *contradiction*
**next**
**fix** *evsR3 S A U s a b c d n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**assume** *B*: *IntAgrK* (*S* (*Card n′*, *n′*, *run′*)) = *None*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(⦇*IntAgrK* := *Some d*, *ExtAgrK* := *Some* (*c* ∗ (*s* + *a* ∗ *b*))⦈)
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**proof** (*rule disjE* [*OF A*], *rule disjI1*, *simp add*: *fun-upd-apply*, *rule impI*, *simp add*: *B*)
**qed** (*rule disjI2*, *simp add*: *fun-upd-apply*)
**next**
**fix** *evsR3 A U s s′ a b c d n′ run′ X* **and** *S* :: *state*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(⦇*IntAgrK* := *Some d*, *ExtAgrK* := *Some* (*c* ∗ (*s′* + *a* ∗ *b*))⦈)
**assume** (*evsR3*, *S*, *A*, *U*) ∈ *protocol*
**hence** *A* ⊆ *U*

**by** (*rule pr-analz-used*)
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *A*
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U* **..**
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *?P ?S n′ run′* ∨
(∃ *b*. *Pri-AgrK b* ∈ *A* ∧ *ExtMapK* (*?S* (*X*, *n′*, *run′*)) = *Some b*)
**by** *contradiction*
**next**
**fix** *evsR3 S A U s a b c d n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**assume** *B*: *IntAgrK* (*S* (*Card n′*, *n′*, *run′*)) = *None*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(|*IntAgrK* := *Some d*, *ExtAgrK* := *Some* (*c* ∗ (*s* + *a* ∗ *b*))|))
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**proof** (*rule disjE* [*OF A*], *rule disjI1*, *simp add*: *fun-upd-apply*, *rule impI*, *simp add*: *B*)
**qed** (*rule disjI2*, *simp add*: *fun-upd-apply*)
**next**
**fix** *evsFR3 A U s a b c d n′ run′* **and** *S* :: *state*
**assume** *sesK K* = *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))
**with** *sesK-inj* **have** *K* = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*)
**by** (*rule injD*)
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately have** *Key* (*sesK K*) ∉ *U*
**by** *simp*
**moreover assume**
(*evsFR3*, *S*, *A*, *U*) ∈ *protocol* **and**
{|*Key* (*sesK K*), *Agent* (*User m*), *Number n*, *Number run*|} ∈ *U*
**hence** *Key* (*sesK K*) ∈ *U*
**by** (*rule pr-sesk-user-2*)
**ultimately show** *?P S n run* ∨ *?Q S A n run*
**by** *contradiction*
**next**
**fix** *evsC4 S A U f m′ n′ run′*
**assume**
*?H1 U* ⟶ *?H2 A* ⟶ *?P S n run* ∨ *?Q S A n run* **and**
*?H1 U* **and**
*?H2 A*
**hence** *A*: *?P S n run* ∨ *?Q S A n run*
**by** *simp*
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*ExtAgrK* := *Some f*|))
**show** *?P ?S n run* ∨ *?Q ?S A n run*
**by** (*rule disjE* [*OF A*], *simp-all add*: *fun-upd-apply*, *blast*)

**qed**

**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** (*erule protocol.induct*, *subst parts-simp*, (*simp*, *blast*)+,
*simp-all add*: *parts-simp-insert parts-auth-data parts-crypt parts-mpair*,
*rule-tac* [!] *impI*)
**fix** *evsR4 S A U n run d e*
**assume**
(*evsR4*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**thus** *Key* (*sesK* (*d* ∗ *e*)) ∈ *U*
**by** (*rule pr-sesk-card*)
**next**
**fix** *evsFR4 S A U m n run c f*
**assume** *A*: (*evsFR4*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* **and**
*ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f*
**hence** {|*Key* (*sesK* (*c* ∗ *f*)), *Agent* (*User m*), *Number n*, *Number run*|} ∈ *U*
**by** (*rule pr-sesk-user-1*)
**with** *A* **show** *Key* (*sesK* (*c* ∗ *f*)) ∈ *U*
**by** (*rule pr-sesk-user-2*)
**qed**

**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** (*erule protocol.induct*, *simp-all*, *rule-tac* [!] *impI*)
**fix** *evsC5 S A U m n run s a b c f g X*
**assume** (*evsC5*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *Says n run 4 X* (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*))
{|*pubAK* (*c* ∗ (*s* + *a* ∗ *b*)), *Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}) ∈ *set evsC5*
(**is** *Says - - - - - ?M* ∈ -)
**hence** *?M* ∈ *spies evsC5*
**by** (*rule set-spies*)
**hence** *?M* ∈ *A* ∪ *spies evsC5* **..**
**hence** *?M* ∈ *parts* (*A* ∪ *spies evsC5*)
**by** (*rule parts.Inj*)
**ultimately show** *Key* (*sesK* (*c* ∗ *f*)) ∈ *U*
**by** (*rule pr-sesk-auth*)

**next**
**fix** *evsFC5 S A U n run d e*
**assume**
(*evsFC5*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**thus** *Key* (*sesK* (*d* ∗ *e*)) ∈ *U*
**by** (*rule pr-sesk-card*)
**qed**

**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* ⟶
*ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b* ⟶
*IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* ⟶
*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*)) ⟶
*s′* + *a′* ∗ *b′* = *s* + *a* ∗ *b* ⟶
*Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
**apply** (*erule protocol.induct*, *rule-tac* [!] *impI*, *simp-all add*: *pr-pri-agrk-analz*)
**apply** (*rule impI*)+
**apply** (*rule-tac* [*2*] *impI*)
**apply** (*rule-tac* [*2*] *conjI*)
**apply** (*rule-tac* [*2*−*3*] *impI*)+
**apply** (*rule-tac* [*4*] *impI*)+
**apply** (*rule-tac* [*5*] *impI*)+
**apply** (*rule-tac* [*6*] *conjI*, (*rule-tac* [*6*] *impI*)+)+
**apply** (*rule-tac* [*6*] *conjI*)
**apply** (*erule-tac* [*6*] *conjE*)+
**apply** (*rule-tac* [*8*] *impI*)+
**apply** (*rule-tac* [*8*] *notI*)
**apply** (*rule-tac* [*9*] *impI*, *rule-tac* [*9*] *conjI*)+
**apply** (*rule-tac* [*9*] *impI*)+
**apply** (*rule-tac* [*10*] *impI*)+
**apply** (*rule-tac* [*10*] *notI*)
**apply** (*rule-tac* [*11*] *impI*)+
**apply** (*rule-tac* [*12*] *impI*)+
**apply** (*rule-tac* [*12*] *notI*)
**proof** *simp-all*
**fix** *evsR1 S A U n run*
**assume** (*evsR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*NonceS* (*S* (*Card n*, *n*, *run*)) = *None*
**hence** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-1*)

**moreover assume** *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b′*
**ultimately show** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
**by** *simp*
**next**
**fix** *evsC2 S A U m n run*
**assume** *A*: (*evsC2*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *NonceS* (*S* (*User m*, *n*, *run*)) = *None*
**ultimately have** *IntMapK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-1*)
**with** *A* **have** *ExtMapK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-2*)
**moreover assume** *ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b*
**ultimately show** *Key* (*sesK* (*c* ∗ *d* ∗ (*a* ∗ *b*))) ∉ *A*
**by** *simp*
**next**
**fix** *evsC2 S A U m n run*
**assume** *A*: (*evsC2*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *NonceS* (*S* (*User m*, *n*, *run*)) = *None*
**ultimately have** *IntMapK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-1*)
**with** *A* **have** *ExtMapK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-2*)
**moreover assume** *ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b*
**ultimately show** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
**by** *simp*
**next**
**fix** *evsR2 S A U n run*
**assume** *A*: (*evsR2*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**hence** *ExtMapK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-2*)
**with** *A* **have** *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *None*
**by** (*rule pr-state-3*)
**moreover assume** *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d*
**ultimately show** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
**by** *simp*
**next**
**fix** *evsC3 S A U n run*
**assume** *A*: (*evsC3*, *S*, *A*, *U*) ∈ *protocol* **and**
*NonceS* (*S* (*Card n*, *n*, *run*)) = *Some s′* **and**
*IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b′* **and**
*ExtMapK* (*S* (*Card n*, *n*, *run*)) = *Some a′* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d*
**moreover assume** *B*: *s′* + *a′* ∗ *b′* = *s* + *a* ∗ *b* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some* (*c* ∗ (*s* + *a* ∗ *b*))
**hence** *ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some* (*c* ∗ (*s′* + *a′* ∗ *b′*))
**by** *simp*
**moreover assume** *C*: *Pri-AgrK c* ∉ *U*
**have** *A* ⊆ *U*

**using** $A$ **by** (*rule pr-analz-used*)
**hence** *Pri-AgrK* $c \notin A$
**using** $C$ **by** (*rule contra-subsetD*)
**ultimately have** *Key* (*sesK* $(c * d * (s' + a' * b'))) \notin A$
**by** (*rule pr-ext-agrk-card*)
**thus** *Key* (*sesK* $(c * d * (s + a * b))) \notin A$
**using** $B$ **by** *simp*
**next**
**fix** *evsR3* $S$ $A$ $U$ $n$ *run*
**assume** (*evsR3*, $S$, $A$, $U$) $\in$ *protocol*
**moreover assume** $0 < m$
**hence** *User* $m \neq$ *Spy*
**by** *simp*
**moreover assume** *IntAgrK* (*S* (*User* $m$, $n$, *run*)) = *Some* $c$
**ultimately have** *Pri-AgrK* $c \notin A$
**by** (*rule pr-int-agrk-user-2*)
**moreover assume** *Pri-AgrK* $c \in A$
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsR3* $S$ $A$ $U$
**assume** (*evsR3*, $S$, $A$, $U$) $\in$ *protocol*
**hence** $A \subseteq U$
**by** (*rule pr-analz-used*)
**moreover assume** *Key* (*sesK* $(c * d * (s + a * b))) \notin U$
**ultimately show** *Key* (*sesK* $(c * d * (s + a * b))) \notin A$
**by** (*rule contra-subsetD*)
**next**
**fix** *evsR3* $S$ $A$ $U$ $s'$ $a'$ $b'$ $c'$ $d'$
**assume**
(*evsR3*, $S$, $A$, $U$) $\in$ *protocol* **and**
*IntAgrK* (*S* (*Card* $n$, $n$, *run*)) = *Some* $d$ **and**
*ExtAgrK* (*S* (*Card* $n$, $n$, *run*)) = *Some* $(c * (s + a * b))$
**hence** *Key* (*sesK* $(d * (c * (s + a * b))))) \in U$
**by** (*rule pr-sesk-card*)
**moreover have** $d * (c * (s + a * b)) = c * d * (s + a * b)$
**by** *simp*
**ultimately have** *Key* (*sesK* $(c * d * (s + a * b))) \in U$
**by** *simp*
**moreover assume** *sesK* $(c * d * (s + a * b))$ = *sesK* $(c' * d' * (s' + a' * b'))$
**ultimately have** *Key* (*sesK* $(c' * d' * (s' + a' * b'))) \in U$
**by** *simp*
**moreover assume** *Key* (*sesK* $(c' * d' * (s' + a' * b'))) \notin U$
**ultimately show** *False*
**by** *simp*
**next**
**fix** *evsR3* $S$ $A$ $U$ $s'$ $a'$ $b'$ $c'$ $d'$
**assume**
(*evsR3*, $S$, $A$, $U$) $\in$ *protocol* **and**

*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some* (*c* ∗ (*s* + *a* ∗ *b*))
**hence** *Key* (*sesK* (*d* ∗ (*c* ∗ (*s* + *a* ∗ *b*)))) ∈ *U*
**by** (*rule pr-sesk-card*)
**moreover have** *d* ∗ (*c* ∗ (*s* + *a* ∗ *b*)) = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*)
**by** *simp*
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** *simp*
**moreover assume** *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*)) = *sesK* (*c*′ ∗ *d*′ ∗ (*s*′ + *a*′ ∗ *b*′))
**ultimately have** *Key* (*sesK* (*c*′ ∗ *d*′ ∗ (*s*′ + *a*′ ∗ *b*′))) ∈ *U*
**by** *simp*
**moreover assume** *Key* (*sesK* (*c*′ ∗ *d*′ ∗ (*s*′ + *a*′ ∗ *b*′))) ∉ *U*
**ultimately show** *False*
**by** *simp*
**next**
**fix** *evsR3 S A U s*′ *c*′
**assume** (*evsR3*, *S*, *A*, *U*) ∈ *protocol*
**hence** *A* ⊆ *U*
**by** (*rule pr-analz-used*)
**moreover assume** *Key* (*sesK* (*c*′ ∗ *d* ∗ (*s*′ + *a*′ ∗ *b*′))) ∉ *U*
**ultimately have** *Key* (*sesK* (*c*′ ∗ *d* ∗ (*s*′ + *a*′ ∗ *b*′))) ∉ *A*
**by** (*rule contra-subsetD*)
**moreover assume** *c*′ ∗ (*s*′ + *a*′ ∗ *b*′) = *c* ∗ (*s* + *a* ∗ *b*)
**hence** *c*′ ∗ *d* ∗ (*s*′ + *a*′ ∗ *b*′) = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*)
**by** *simp*
**ultimately show** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *A*
**by** *simp*
**next**
**fix** *evsFR3 S A U s*′ *a*′ *b*′ *c*′ *d*′
**assume**
(*evsFR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some* (*c* ∗ (*s* + *a* ∗ *b*))
**hence** *Key* (*sesK* (*d* ∗ (*c* ∗ (*s* + *a* ∗ *b*)))) ∈ *U*
**by** (*rule pr-sesk-card*)
**moreover have** *d* ∗ (*c* ∗ (*s* + *a* ∗ *b*)) = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*)
**by** *simp*
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** *simp*
**moreover assume** *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*)) = *sesK* (*c*′ ∗ *d*′ ∗ (*s*′ + *a*′ ∗ *b*′))
**ultimately have** *Key* (*sesK* (*c*′ ∗ *d*′ ∗ (*s*′ + *a*′ ∗ *b*′))) ∈ *U*
**by** *simp*
**moreover assume** *Key* (*sesK* (*c*′ ∗ *d*′ ∗ (*s*′ + *a*′ ∗ *b*′))) ∉ *U*
**ultimately show** *False*
**by** *simp*
**qed**

**lemma** *pr-sign-key-used*:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹ *Key* (*priSK X*) ∈ *U*

**by** (*erule protocol.induct*, *simp-all*)

**lemma** *pr-sign-key-analz*:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹ *Key* (*priSK X*) ∉ *analz* (*A* ∪ *spies evs*)
**proof** (*simp add*: *pr-key-analz*, *erule protocol.induct*,
*auto simp add*: *priSK-pubSK priSK-symK*)
**fix** *evsR3 S A U s a b c d*
**assume** (*evsR3*, *S*, *A*, *U*) ∈ *protocol*
**hence** *Key* (*priSK X*) ∈ *U*
**by** (*rule pr-sign-key-used*)
**moreover assume** *priSK X* = *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** *simp*
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsFR3 S A U s a b c d*
**assume** (*evsFR3*, *S*, *A*, *U*) ∈ *protocol*
**hence** *Key* (*priSK X*) ∈ *U*
**by** (*rule pr-sign-key-used*)
**moreover assume** *priSK X* = *sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** *simp*
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**qed**

**lemma** *pr-auth-data-parts* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*Auth-Data* (*priAK n*) *b* ∈ *parts* (*A* ∪ *spies evs*) ⟶
(∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*)
(**is** - ⟹ *?M* ∈ - ⟶ -)
**apply** (*erule protocol.induct*, *simp*, *subst parts-simp*, *simp*, *blast*+, *simp-all*
*add*: *parts-simp-insert parts-auth-data parts-crypt parts-mpair del*: *fun-upd-apply*)
**apply** (*rule impI*)
**apply** ((*rule-tac* [*2*] *conjI*)*?*, *rule-tac* [*2*] *impI*)+
**apply** (*rule-tac* [*3*] *impI*)+
**apply** (*rule-tac* [*4*] *impI*, (*rule-tac* [*4*] *conjI*)*?*)+
**apply** (*rule-tac* [*5*] *impI*)+
**apply** (*rule-tac* [*6*] *impI*, (*rule-tac* [*6*] *conjI*)*?*)+
**apply** (*rule-tac* [*7*] *impI*)+
**apply** (*rule-tac* [*8*] *impI*, (*rule-tac* [*8*] *conjI*)*?*)+
**apply** (*rule-tac* [*9*] *impI*)+
**apply** (*rule-tac* [*10*] *impI*)
**apply** (*rule-tac* [*11*] *conjI*)
**apply** (*rule-tac* [*11*−*12*] *impI*)+
**apply** (*rule-tac* [*13*] *conjI*)

```
            apply (rule-tac [13-14] impI)+
            apply (rule-tac [15-17] impI)
            apply (erule-tac [17] conjE)
proof -
  fix evsR1 A n' run' s and S :: state
  let ?S = S((Card n', n', run') := S (Card n', n', run')
    (|NonceS := Some s|))
  assume
   ?M ∈ parts (A ∪ spies evsR1) ⟶
     (∃ m run. IntMapK (S (Card m, m, run)) = Some b) and
   ?M ∈ parts (A ∪ spies evsR1)
  hence ∃ m run. IntMapK (S (Card m, m, run)) = Some b ..
  then obtain m and run where IntMapK (S (Card m, m, run)) = Some b
   by blast
  thus ∃ m run. IntMapK (?S (Card m, m, run)) = Some b
   by (rule-tac x = m in exI, rule-tac x = run in exI, simp, blast)
next
  fix evsC2 A n' run' s a and S :: state
  let ?S = S((Spy, n', run') := S (Spy, n', run')
    (|NonceS := Some s, IntMapK := Some a|))
  assume
   ?M ∈ parts (A ∪ spies evsC2) ⟶
     (∃ m run. IntMapK (S (Card m, m, run)) = Some b) and
   ?M ∈ parts (A ∪ spies evsC2)
  hence ∃ m run. IntMapK (S (Card m, m, run)) = Some b ..
  then obtain m and run where IntMapK (S (Card m, m, run)) = Some b
   by blast
  thus ∃ m run. IntMapK (?S (Card m, m, run)) = Some b
   by (rule-tac x = m in exI, rule-tac x = run in exI, simp)
next
  fix evsC2 A n' run' m' s a and S :: state
  let ?S = S((User m', n', run') := S (User m', n', run')
    (|NonceS := Some s, IntMapK := Some a|))
  assume
   ?M ∈ parts (A ∪ spies evsC2) ⟶
     (∃ m run. IntMapK (S (Card m, m, run)) = Some b) and
   ?M ∈ parts (A ∪ spies evsC2)
  hence ∃ m run. IntMapK (S (Card m, m, run)) = Some b ..
  then obtain m and run where IntMapK (S (Card m, m, run)) = Some b
   by blast
  thus ∃ m run. IntMapK (?S (Card m, m, run)) = Some b
   by (rule-tac x = m in exI, rule-tac x = run in exI, simp)
next
  fix evsC2 A n' run' s a and S :: state
  let ?S = S((Spy, n', run') := S (Spy, n', run')
    (|NonceS := Some s, IntMapK := Some a|))
  assume
   ?M ∈ parts (A ∪ spies evsC2) ⟶
     (∃ m run. IntMapK (S (Card m, m, run)) = Some b) and
```

*?M* ∈ *parts* (*A* ∪ *spies evsC2*)
**hence** ∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b* **..**
**then obtain** *m* **and** *run* **where** *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*
**by** *blast*
**thus** ∃ *m run*. *IntMapK* (*?S* (*Card m*, *m*, *run*)) = *Some b*
**by** (*rule-tac x* = *m* **in** *exI*, *rule-tac x* = *run* **in** *exI*, *simp*)
**next**
**fix** *evsC2 A n′ run′ m′ a* **and** *S* :: *state*
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*NonceS* := *Some* 0, *IntMapK* := *Some a*|))
**assume**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*) ⟶
(∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*) **and**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*)
**hence** ∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b* **..**
**then obtain** *m* **and** *run* **where** *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*
**by** *blast*
**thus** ∃ *m run*. *IntMapK* (*?S* (*Card m*, *m*, *run*)) = *Some b*
**by** (*rule-tac x* = *m* **in** *exI*, *rule-tac x* = *run* **in** *exI*, *simp*)
**next**
**fix** *evsC2 A n′ run′ s a* **and** *S* :: *state*
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**assume**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*) ⟶
(∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*) **and**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*)
**hence** ∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b* **..**
**then obtain** *m* **and** *run* **where** *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*
**by** *blast*
**thus** ∃ *m run*. *IntMapK* (*?S* (*Card m*, *m*, *run*)) = *Some b*
**by** (*rule-tac x* = *m* **in** *exI*, *rule-tac x* = *run* **in** *exI*, *simp*)
**next**
**fix** *evsC2 A n′ run′ m′ s a* **and** *S* :: *state*
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*NonceS* := *Some s*, *IntMapK* := *Some a*|))
**assume**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*) ⟶
(∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*) **and**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*)
**hence** ∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b* **..**
**then obtain** *m* **and** *run* **where** *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*
**by** *blast*
**thus** ∃ *m run*. *IntMapK* (*?S* (*Card m*, *m*, *run*)) = *Some b*
**by** (*rule-tac x* = *m* **in** *exI*, *rule-tac x* = *run* **in** *exI*, *simp*)
**next**
**fix** *evsC2 A n′ run′ a* **and** *S* :: *state*
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*NonceS* := *Some* 0, *IntMapK* := *Some a*|))

**assume**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*) ⟶
(∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*) **and**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*)
**hence** ∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b* **..**
**then obtain** *m* **and** *run* **where** *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*
**by** *blast*
**thus** ∃ *m run*. *IntMapK* (*?S* (*Card m*, *m*, *run*)) = *Some b*
**by** (*rule-tac x* = *m* **in** *exI*, *rule-tac x* = *run* **in** *exI*, *simp*)
**next**
**fix** *evsC2 A n′ run′ m′ a* **and** *S* :: *state*
**let** *?S* = *S*((*User m′*, *n′*, *run′*) := *S* (*User m′*, *n′*, *run′*)
(|*NonceS* := *Some* 0, *IntMapK* := *Some a*|))
**assume**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*) ⟶
(∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*) **and**
*?M* ∈ *parts* (*A* ∪ *spies evsC2*)
**hence** ∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b* **..**
**then obtain** *m* **and** *run* **where** *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*
**by** *blast*
**thus** ∃ *m run*. *IntMapK* (*?S* (*Card m*, *m*, *run*)) = *Some b*
**by** (*rule-tac x* = *m* **in** *exI*, *rule-tac x* = *run* **in** *exI*, *simp*)
**next**
**fix** *evsR2 A n′ run′ a b′* **and** *S* :: *state*
**let** *?S* = *S*((*Card n′*, *n′*, *run′*) := *S* (*Card n′*, *n′*, *run′*)
(|*IntMapK* := *Some b′*, *ExtMapK* := *Some a*|))
**assume**
*?M* ∈ *parts* (*A* ∪ *spies evsR2*) ⟶
(∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*) **and**
*?M* ∈ *parts* (*A* ∪ *spies evsR2*)
**hence** ∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b* **..**
**then obtain** *m* **and** *run* **where** *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*
**by** *blast*
**moreover assume** *IntMapK* (*S* (*Card n′*, *n′*, *run′*)) = *None*
**ultimately show** ∃ *m run*. *IntMapK* (*?S* (*Card m*, *m*, *run*)) = *Some b*
**proof** (*rule-tac x* = *m* **in** *exI*, *rule-tac x* = *run* **in** *exI*, *simp*)
**qed** (*rule impI*, *simp*)
**next**
**fix** *evsC3 A n′ run′ b′ c* **and** *S* :: *state*
**let** *?S* = *S*((*Spy*, *n′*, *run′*) := *S* (*Spy*, *n′*, *run′*)
(|*ExtMapK* := *Some b′*, *IntAgrK* := *Some c*|))
**assume**
*?M* ∈ *parts* (*A* ∪ *spies evsC3*) ⟶
(∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*) **and**
*?M* ∈ *parts* (*A* ∪ *spies evsC3*)
**hence** ∃ *m run*. *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b* **..**
**then obtain** *m* **and** *run* **where** *IntMapK* (*S* (*Card m*, *m*, *run*)) = *Some b*
**by** *blast*
**thus** ∃ *m run*. *IntMapK* (*?S* (*Card m*, *m*, *run*)) = *Some b*

**by** (*rule-tac* $x = m$ **in** *exI*, *rule-tac* $x = run$ **in** *exI*, *simp*)
**next**
**fix** *evsC3 A n′ run′ b′ c m* **and** $S$ :: *state*
**let** *?S* = $S((User\ m,\ n',\ run') := S\ (User\ m,\ n',\ run')$
$(\!|ExtMapK := Some\ b',\ IntAgrK := Some\ c|\!)$)
**assume**
*?M* ∈ *parts* $(A \cup spies\ evsC3) \longrightarrow$
$(\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b)$ **and**
*?M* ∈ *parts* $(A \cup spies\ evsC3)$
**hence** $\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b$ **..**
**then obtain** $m$ **and** *run* **where** $IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b$
**by** *blast*
**thus** $\exists\, m\ run.\ IntMapK\ (?S\ (Card\ m,\ m,\ run)) = Some\ b$
**by** (*rule-tac* $x = m$ **in** *exI*, *rule-tac* $x = run$ **in** *exI*, *simp*)
**next**
**fix** *evsR3 A n′ run′ s a b′ c d* **and** $S$ :: *state*
**let** *?S* = $S((Card\ n',\ n',\ run') := S\ (Card\ n',\ n',\ run')$
$(\!|IntAgrK := Some\ d,\ ExtAgrK := Some\ (c * (s + a * b'))|\!)$)
**assume**
*?M* ∈ *parts* $(A \cup spies\ evsR3) \longrightarrow$
$(\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b)$ **and**
*?M* ∈ *parts* $(A \cup spies\ evsR3)$
**hence** $\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b$ **..**
**then obtain** $m$ **and** *run* **where** $IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b$
**by** *blast*
**thus** $\exists\, m\ run.\ IntMapK\ (?S\ (Card\ m,\ m,\ run)) = Some\ b$
**by** (*rule-tac* $x = m$ **in** *exI*, *rule-tac* $x = run$ **in** *exI*, *simp*, *blast*)
**next**
**fix** *evsR3 A n′ run′ s a b′ c d* **and** $S$ :: *state*
**let** *?S* = $S((Card\ n',\ n',\ run') := S\ (Card\ n',\ n',\ run')$
$(\!|IntAgrK := Some\ d,\ ExtAgrK := Some\ (c * (s + a * b'))|\!)$)
**assume**
*?M* ∈ *parts* $(A \cup spies\ evsR3) \longrightarrow$
$(\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b)$ **and**
*?M* ∈ *parts* $(A \cup spies\ evsR3)$
**hence** $\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b$ **..**
**then obtain** $m$ **and** *run* **where** $IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b$
**by** *blast*
**thus** $\exists\, m\ run.\ IntMapK\ (?S\ (Card\ m,\ m,\ run)) = Some\ b$
**by** (*rule-tac* $x = m$ **in** *exI*, *rule-tac* $x = run$ **in** *exI*, *simp*, *blast*)
**next**
**fix** *evsC4 A n′ run′ c f m* **and** $S$ :: *state*
**let** *?S* = $S((User\ m,\ n',\ run') := S\ (User\ m,\ n',\ run')(\!|ExtAgrK := Some\ f|\!))$
**assume**
*?M* ∈ *parts* $(A \cup spies\ evsC4) \longrightarrow$
$(\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b)$ **and**
*?M* ∈ *parts* $(A \cup spies\ evsC4)$
**hence** $\exists\, m\ run.\ IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b$ **..**
**then obtain** $m$ **and** *run* **where** $IntMapK\ (S\ (Card\ m,\ m,\ run)) = Some\ b$

```
  by blast
 thus ∃ m run. IntMapK (?S (Card m, m, run)) = Some b
  by (rule-tac x = m in exI, rule-tac x = run in exI, simp)
next
 fix evsR4 n' run' b' and S :: state
 assume IntMapK (S (Card n', n', run')) = Some b'
 thus ∃ m run. IntMapK (S (Card m, m, run)) = Some b'
  by blast
next
 fix evsFR4 A U s a b' c f g and S :: state
 assume
   A: ?M ∈ parts (A ∪ spies evsFR4) ⟶
     (∃ m run. IntMapK (S (Card m, m, run)) = Some b) and
   B: (evsFR4, S, A, U) ∈ protocol and
   C: Crypt (sesK (c ∗ f))
     {|pubAK (c ∗ (s + a ∗ b')), Auth-Data g b', pubAK g,
      Crypt (priSK CA) (Hash (pubAK g))|} ∈ synth (analz (A ∪ spies evsFR4))
     (is Crypt - ?M' ∈ synth (analz ?A)) and
   D: priAK n = g and
   E: b = b'
 show ∃ m run. IntMapK (S (Card m, m, run)) = Some b'
 proof −
   have Crypt (sesK (c ∗ f)) ?M' ∈ analz ?A ∨
     ?M' ∈ synth (analz ?A) ∧ Key (sesK (c ∗ f)) ∈ analz ?A
    using C by (rule synth-crypt)
   moreover {
     assume Crypt (sesK (c ∗ f)) ?M' ∈ analz ?A
     hence Crypt (sesK (c ∗ f)) ?M' ∈ parts ?A
      by (rule subsetD [OF analz-parts-subset])
     hence ?M' ∈ parts ?A
      by (rule parts.Body)
     hence {|Auth-Data g b', pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
       ∈ parts ?A
      by (rule parts.Snd)
     hence Auth-Data g b' ∈ parts ?A
      by (rule parts.Fst)
   }
   moreover {
     assume ?M' ∈ synth (analz ?A) ∧ Key (sesK (c ∗ f)) ∈ analz ?A
     hence ?M' ∈ synth (analz ?A) ..
     hence {|Auth-Data g b', pubAK g, Crypt (priSK CA) (Hash (pubAK g))|}
       ∈ synth (analz ?A)
      by (rule synth-analz-snd)
     hence Auth-Data g b' ∈ synth (analz ?A)
      by (rule synth-analz-fst)
     hence Auth-Data g b' ∈ analz ?A ∨
       Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b' ∈ analz ?A
      by (rule synth-auth-data)
     moreover {
```

```
      assume Auth-Data g b' ∈ analz ?A
      hence Auth-Data g b' ∈ parts ?A
       by (rule subsetD [OF analz-parts-subset])
     }
     moreover {
      assume Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b' ∈ analz ?A
      hence Pri-AgrK g ∈ analz ?A ..
      hence Pri-AgrK (priAK n) ∈ analz ?A
       using D by simp
      moreover have Pri-AgrK (priAK n) ∉ analz ?A
       using B by (rule pr-auth-key-analz)
      ultimately have Auth-Data g b' ∈ parts ?A
       by contradiction
     }
     ultimately have Auth-Data g b' ∈ parts ?A ..
    }
    ultimately have Auth-Data g b' ∈ parts ?A ..
    thus ?thesis
     using A and D and E by simp
  qed
qed

lemma pr-sign-parts [rule-format]:
 (evs, S, A, U) ∈ protocol ⟹
    Crypt (priSK CA) (Hash (pubAK g)) ∈ parts (A ∪ spies evs) ⟶
  (∃ n. g = priAK n)
  (is - ⟹ ?M g ∈ - ⟶ -)
proof (erule protocol.induct, subst parts-simp, (simp, blast)+, simp-all add:
 parts-simp-insert parts-auth-data parts-crypt parts-mpair, rule-tac [!] impI)
  fix evsR4 n
  assume agrK g = agrK (priAK n)
  with agrK-inj have g = priAK n
   by (rule injD)
  thus ∃ n. g = priAK n ..
next
  fix evsFR4 S A U s a b c f g'
  assume
    A: ?M g ∈ parts (A ∪ spies evsFR4) ⟶ (∃ n. g = priAK n) and
    B: (evsFR4, S, A, U) ∈ protocol and
    C: Crypt (sesK (c * f)) {|pubAK (c * (s + a * b)), Auth-Data g' b,
      pubAK g', ?M g'|} ∈ synth (analz (A ∪ spies evsFR4))
      (is ?M' ∈ synth (analz ?A))
  assume agrK g = agrK g'
  with agrK-inj have D: g = g'
   by (rule injD)
  show ∃ n. g = priAK n
  proof -
    have ?M' ∈ analz ?A ∨
      {|pubAK (c * (s + a * b)), Auth-Data g' b, pubAK g', ?M g'|}
```

```
    ∈ synth (analz ?A) ∧
   Key (sesK (c ∗ f)) ∈ analz ?A
  using C by (rule synth-crypt)
 moreover {
  assume ?M′ ∈ analz ?A
  hence ?M′ ∈ parts ?A
   by (rule subsetD [OF analz-parts-subset])
  hence {|pubAK (c ∗ (s + a ∗ b)), Auth-Data g′ b, pubAK g′, ?M g′|}
    ∈ parts ?A
   by (rule parts.Body)
  hence ?M g′ ∈ parts ?A
   by (rule-tac parts.Snd, assumption?)+
  hence ∃ n. g′ = priAK n
   using A and D by simp
 }
 moreover {
  assume
   {|pubAK (c ∗ (s + a ∗ b)), Auth-Data g′ b, pubAK g′, ?M g′|}
     ∈ synth (analz ?A) ∧
    Key (sesK (c ∗ f)) ∈ analz ?A
  hence
   {|pubAK (c ∗ (s + a ∗ b)), Auth-Data g′ b, pubAK g′, ?M g′|}
     ∈ synth (analz ?A) ..
  hence {|Auth-Data g′ b, pubAK g′, ?M g′|} ∈ synth (analz ?A)
   by (rule synth-analz-snd)
  hence {|pubAK g′, ?M g′|} ∈ synth (analz ?A)
   by (rule synth-analz-snd)
  hence ?M g′ ∈ synth (analz ?A)
   by (rule synth-analz-snd)
  hence ?M g′ ∈ analz ?A ∨
    Hash (pubAK g′) ∈ synth (analz ?A) ∧ Key (priSK CA) ∈ analz ?A
   by (rule synth-crypt)
  moreover {
   assume ?M g′ ∈ analz ?A
   hence ?M g′ ∈ parts ?A
    by (rule subsetD [OF analz-parts-subset])
   hence ∃ n. g′ = priAK n
    using A and D by simp
  }
  moreover {
   assume Hash (pubAK g′) ∈ synth (analz ?A) ∧
     Key (priSK CA) ∈ analz ?A
   hence Key (priSK CA) ∈ analz ?A ..
   moreover have Key (priSK CA) ∉ analz ?A
    using B by (rule pr-sign-key-analz)
   ultimately have ∃ n. g′ = priAK n
    by contradiction
  }
  ultimately have ∃ n. g′ = priAK n ..
```

```
    }
    ultimately have ∃ n. g' = priAK n ..
    thus ∃ n. g = priAK n
     using D by simp
  qed
qed

lemma pr-key-secrecy-aux [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 ⟶
    ExtMapK (S (User m, n, run)) = Some b ⟶
    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-Data g b, pubAK g,
       Crypt (priSK CA) (Hash (pubAK g))|}) ∈ set evs ⟶
  Key (sesK (c * f)) ∉ A
supply [[simproc del: defined-all]]
apply (erule protocol.induct, (rule-tac [!] impI)+, simp-all split: if-split-asm)
       apply (erule-tac [7] disjE)
        apply simp-all
       apply (erule-tac [7] conjE)+
       apply (erule-tac [8] disjE)+
        apply (erule-tac [8] conjE)+
        apply simp-all
       apply (rule-tac [8] notI)
proof −
  fix evsC2 S A U m n run
  assume A: (evsC2, S, A, U) ∈ protocol
  moreover assume NonceS (S (User m, n, run)) = None
  ultimately have IntMapK (S (User m, n, run)) = None
   by (rule pr-state-1)
  with A have ExtMapK (S (User m, n, run)) = None
   by (rule pr-state-2)
  moreover assume ExtMapK (S (User m, n, run)) = Some b
  ultimately show Key (sesK (c * f)) ∉ A
   by simp
next
  fix evsC2 S A U m n run
  assume A: (evsC2, S, A, U) ∈ protocol
  moreover assume NonceS (S (User m, n, run)) = None
  ultimately have IntMapK (S (User m, n, run)) = None
   by (rule pr-state-1)
  with A have ExtMapK (S (User m, n, run)) = None
   by (rule pr-state-2)
  moreover assume ExtMapK (S (User m, n, run)) = Some b
  ultimately show Key (sesK (c * f)) ∉ A
```

**by** *simp*
**next**
**fix** *evsC3 S A U m n run*
**assume** *A*: (*evsC3*, *S*, *A*, *U*) ∈ *protocol* **and**
*ExtMapK* (*S* (*User m*, *n*, *run*)) = *None*
**hence** *IntAgrK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-3*)
**with** *A* **have** *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *None*
**by** (*rule pr-state-4*)
**moreover assume** *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f*
**ultimately show** *Key* (*sesK* (*c* ∗ *f*)) ∉ *A*
**by** *simp*
**next**
**fix** *evsR3 S A U d s′ s″ a′ b′ c′*
**assume**
*A*: (*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*B*: *Key* (*sesK* (*c′* ∗ *d* ∗ (*s″* + *a′* ∗ *b′*))) ∉ *U* **and**
*C*: *Says n run 4 X* (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*))
{|*pubAK* (*c* ∗ (*s* + *a* ∗ *b*)), *Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}) ∈ *set evsR3*
(**is** *Says - - - - - ?M* ∈ -)
**show** *s″* = *s′* ∧ *Pri-AgrK c′* ∈ *analz* (*A* ∪ *spies evsR3*) ⟶
*sesK* (*c* ∗ *f*) ≠ *sesK* (*c′* ∗ *d* ∗ (*s′* + *a′* ∗ *b′*))
**proof** (*rule impI*, *rule notI*, *erule conjE*)
**have** *?M* ∈ *spies evsR3*
**using** *C* **by** (*rule set-spies*)
**hence** *?M* ∈ *A* ∪ *spies evsR3*
**by** *simp*
**hence** *?M* ∈ *parts* (*A* ∪ *spies evsR3*)
**by** (*rule parts.Inj*)
**with** *A* **have** *Key* (*sesK* (*c* ∗ *f*)) ∈ *U*
**by** (*rule pr-sesk-auth*)
**moreover assume** *sesK* (*c* ∗ *f*) = *sesK* (*c′* ∗ *d* ∗ (*s′* + *a′* ∗ *b′*)) **and** *s″* = *s′*
**hence** *Key* (*sesK* (*c* ∗ *f*)) ∉ *U*
**using** *B* **by** *simp*
**ultimately show** *False*
**by** *contradiction*
**qed**
**next**
**fix** *evsFR3 S A U s′ a′ b′ c′ d*
**assume**
*A*: (*evsFR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*B*: *Key* (*sesK* (*c′* ∗ *d* ∗ (*s′* + *a′* ∗ *b′*))) ∉ *U* **and**
*C*: *Says n run 4 X* (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*))
{|*pubAK* (*c* ∗ (*s* + *a* ∗ *b*)), *Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}) ∈ *set evsFR3*
(**is** *Says - - - - - ?M* ∈ -)
**show** *sesK* (*c* ∗ *f*) ≠ *sesK* (*c′* ∗ *d* ∗ (*s′* + *a′* ∗ *b′*))
**proof**

**have** *?M* ∈ *spies evsFR3*
**using** *C* **by** (*rule set-spies*)
**hence** *?M* ∈ *A* ∪ *spies evsFR3*
**by** *simp*
**hence** *?M* ∈ *parts* (*A* ∪ *spies evsFR3*)
**by** (*rule parts.Inj*)
**with** *A* **have** *Key* (*sesK* (*c* ∗ *f*)) ∈ *U*
**by** (*rule pr-sesk-auth*)
**moreover assume** *sesK* (*c* ∗ *f*) = *sesK* (*c*′ ∗ *d* ∗ (*s*′ + *a*′ ∗ *b*′))
**hence** *Key* (*sesK* (*c* ∗ *f*)) ∉ *U*
**using** *B* **by** *simp*
**ultimately show** *False*
**by** *contradiction*
**qed**
**next**
**fix** *evsC4 S A U n run* **and** *m* :: *nat*
**assume** (*evsC4*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *0* < *m*
**hence** *User m* ≠ *Spy*
**by** *simp*
**moreover assume** *Says n run 4 X* (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*))
{|*pubAK* (*c* ∗ (*s* + *a* ∗ *b*)), *Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}) ∈ *set evsC4*
**ultimately have** *ExtAgrK* (*S* (*User m*, *n*, *run*)) ≠ *None*
**by** (*rule pr-ext-agrk-user-2*)
**moreover assume** *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *None*
**ultimately show** *Key* (*sesK* (*c* ∗ *f*)) ∉ *A*
**by** *contradiction*
**next**
**fix** *evsR4 A U n run X s*′ *a*′ *b*′ *d e* **and** *S* :: *state*
**assume** *A*: *User m* = *X*
**assume** *agrK* (*c* ∗ (*s* + *a* ∗ *b*′)) = *agrK e*
**with** *agrK-inj* **have** *B*: *c* ∗ (*s* + *a* ∗ *b*′) = *e*
**by** (*rule injD*)
**assume** *sesK* (*c* ∗ *f*) = *sesK* (*d* ∗ *e*)
**with** *sesK-inj* **have** *c* ∗ *f* = *d* ∗ *e*
**by** (*rule injD*)
**hence** *C*: *c* ∗ *f* = *c* ∗ *d* ∗ (*s* + *a* ∗ *b*′)
**using** *B* **by** *simp*
**assume** *ExtAgrK* (*S* (*X*, *n*, *run*)) = *Some f*
**hence** *D*: *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f*
**using** *A* **by** *simp*
**assume** *E*: (*evsR4*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *0* < *m*
**hence** *F*: *User m* ≠ *Spy*
**by** *simp*
**moreover assume** *NonceS* (*S* (*X*, *n*, *run*)) = *Some s*
**hence** *G*: *NonceS* (*S* (*User m*, *n*, *run*)) = *Some s*
**using** *A* **by** *simp*

**moreover assume** *IntMapK* (*S* (*X*, *n*, *run*)) = *Some a*
**hence** *H*: *IntMapK* (*S* (*User m*, *n*, *run*)) = *Some a*
**using** *A* **by** *simp*
**moreover assume** *ExtMapK* (*S* (*X*, *n*, *run*)) = *Some b′*
**hence** *I*: *ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b′*
**using** *A* **by** *simp*
**moreover assume** *IntAgrK* (*S* (*X*, *n*, *run*)) = *Some c*
**hence** *J*: *IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c*
**using** *A* **by** *simp*
**moreover assume** *K*: *NonceS* (*S* (*Card n*, *n*, *run*)) = *Some s′*
**moreover assume** *L*: *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b′*
**moreover assume** *M*: *ExtMapK* (*S* (*Card n*, *n*, *run*)) = *Some a′*
**moreover assume** *N*: *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d*
**moreover assume** *ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**hence** *ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some* (*c* ∗ (*s* + *a* ∗ *b′*))
**using** *B* **by** *simp*
**moreover assume** *Says n run 4 X* (*Card n*)
(*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK* (*d* ∗ (*s′* + *a′* ∗ *b′*)))) ∈ *set evsR4*
**hence** *Says n run 4* (*User m*) (*Card n*)
(*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK* (*d* ∗ (*s′* + *a′* ∗ *b′*)))) ∈ *set evsR4*
**using** *A* **by** *simp*
**with** *E* **and** *F* **and** *D* **have** *d* ∗ (*s′* + *a′* ∗ *b′*) = *f*
**by** (*rule pr-ext-agrk-user-3*)
**hence** *c* ∗ *d* ∗ (*s′* + *a′* ∗ *b′*) = *c* ∗ *d* ∗ (*s* + *a* ∗ *b′*)
**using** *C* **by** *auto*
**hence** *s′* + *a′* ∗ *b′* = *s* + *a* ∗ *b′*
**proof** *auto*
**assume** *c* = *0*
**moreover have** *c* ∗ (*s* + *a* ∗ *b′*) ≠ *0*
**using** *E* **and** *G* **and** *H* **and** *I* **and** *J* **by** (*rule pr-int-agrk-user-3*)
**ultimately show** *?thesis*
**by** *simp*
**next**
**assume** *d* = *0*
**moreover have** *d* ∗ (*s′* + *a′* ∗ *b′*) ≠ *0*
**using** *E* **and** *K* **and** *L* **and** *M* **and** *N* **by** (*rule pr-int-agrk-card*)
**ultimately show** *?thesis*
**by** *simp*
**qed**
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b′*))) ∉ *A*
**by** (*rule pr-sesk-card-user*)
**moreover have** *c* ∗ *d* ∗ (*s* + *a* ∗ *b′*) = *d* ∗ *e*
**using** *B* **by** *simp*
**ultimately show** *Key* (*sesK* (*d* ∗ *e*)) ∉ *A*
**by** *simp*
**next**
**fix** *evsFR4 S A U m n run b g*
**assume**
*A*: (*evsFR4*, *S*, *A*, *U*) ∈ *protocol* **and**

*B*: *ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b* **and**
*C*: *IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* **and**
*D*: *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f* **and**
*E*: *0* < *m* **and**
*F*: *Key* (*sesK* (*c* ∗ *f*)) ∈ *A*
**assume** *G*: *Crypt* (*sesK* (*c* ∗ *f*))
{|*pubAK* (*c* ∗ (*s* + *a* ∗ *b*)), *Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|} ∈ *synth* (*analz* (*A* ∪ *spies evsFR4*))
(**is** *Crypt* - *?M* ∈ *synth* (*analz ?A*))
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) *?M* ∈ *analz ?A* ∨
*?M* ∈ *synth* (*analz ?A*) ∧ *Key* (*sesK* (*c* ∗ *f*)) ∈ *analz ?A*
**by** (*rule synth-crypt*)
**moreover {**
**assume** *Crypt* (*sesK* (*c* ∗ *f*)) *?M* ∈ *analz ?A*
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) *?M* ∈ *parts ?A*
**by** (*rule subsetD* [*OF analz-parts-subset*])
**hence** *?M* ∈ *parts ?A*
**by** (*rule parts.Body*)
**hence** *Crypt* (*priSK CA*) (*Hash* (*pubAK g*)) ∈ *parts ?A*
**by** (*rule-tac parts.Snd*, *assumption?*)+
**with** *A* **have** ∃ *n*. *g* = *priAK n*
**by** (*rule pr-sign-parts*)
**}**
**moreover {**
**assume** *?M* ∈ *synth* (*analz ?A*) ∧ *Key* (*sesK* (*c* ∗ *f*)) ∈ *analz ?A*
**hence** *?M* ∈ *synth* (*analz ?A*) **..**
**hence** {|*Auth-Data g b*, *pubAK g*, *Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}
∈ *synth* (*analz ?A*)
**by** (*rule synth-analz-snd*)
**hence** {|*pubAK g*, *Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|} ∈ *synth* (*analz ?A*)
**by** (*rule synth-analz-snd*)
**hence** *Crypt* (*priSK CA*) (*Hash* (*pubAK g*)) ∈ *synth* (*analz ?A*)
**by** (*rule synth-analz-snd*)
**hence** *Crypt* (*priSK CA*) (*Hash* (*pubAK g*)) ∈ *analz ?A* ∨
*Hash* (*pubAK g*) ∈ *synth* (*analz ?A*) ∧ *Key* (*priSK CA*) ∈ *analz ?A*
**by** (*rule synth-crypt*)
**moreover {**
**assume** *Crypt* (*priSK CA*) (*Hash* (*pubAK g*)) ∈ *analz ?A*
**hence** *Crypt* (*priSK CA*) (*Hash* (*pubAK g*)) ∈ *parts ?A*
**by** (*rule subsetD* [*OF analz-parts-subset*])
**with** *A* **have** ∃ *n*. *g* = *priAK n*
**by** (*rule pr-sign-parts*)
**}**
**moreover {**
**assume** *Hash* (*pubAK g*) ∈ *synth* (*analz ?A*) ∧ *Key* (*priSK CA*) ∈ *analz ?A*
**hence** *Key* (*priSK CA*) ∈ *analz ?A* **..**
**moreover have** *Key* (*priSK CA*) ∉ *analz ?A*
**using** *A* **by** (*rule pr-sign-key-analz*)
**ultimately have** ∃ *n*. *g* = *priAK n*

**by** *contradiction*
**}**
**ultimately have** $\exists\, n.\ g = priAK\ n$ **..**
**}**
**ultimately have** $\exists\, n.\ g = priAK\ n$ **..**
**then obtain** $n'$ **where** $g = priAK\ n'$ **..**
**hence** *Crypt* (*sesK* $(c * f)$)
$\{\!|pubAK\ (c * (s + a * b)),\ Auth\text{-}Data\ (priAK\ n')\ b,\ pubAK\ (priAK\ n'),$
*Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK* $n'$)))$|\!\} \in synth\ (analz\ ?A)$
(**is** *Crypt* - *?M* $\in$ -)
**using** *G* **by** *simp*
**hence** *Crypt* (*sesK* $(c * f)$) *?M* $\in$ *analz ?A* $\vee$
*?M* $\in$ *synth* (*analz ?A*) $\wedge$ *Key* (*sesK* $(c * f)$) $\in$ *analz ?A*
**by** (*rule synth-crypt*)
**moreover {**
**assume** *Crypt* (*sesK* $(c * f)$) *?M* $\in$ *analz ?A*
**hence** *Crypt* (*sesK* $(c * f)$) *?M* $\in$ *parts ?A*
**by** (*rule subsetD* [*OF analz-parts-subset*])
**hence** *?M* $\in$ *parts ?A*
**by** (*rule parts.Body*)
**hence** $\{\!|Auth\text{-}Data\ (priAK\ n')\ b,\ pubAK\ (priAK\ n'),$
*Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK* $n'$)))$|\!\} \in parts\ ?A$
**by** (*rule parts.Snd*)
**hence** *Auth-Data* (*priAK* $n'$) $b \in$ *parts ?A*
**by** (*rule parts.Fst*)
**}**
**moreover {**
**assume** *?M* $\in$ *synth* (*analz ?A*) $\wedge$ *Key* (*sesK* $(c * f)$) $\in$ *analz ?A*
**hence** *?M* $\in$ *synth* (*analz ?A*) **..**
**hence** $\{\!|Auth\text{-}Data\ (priAK\ n')\ b,\ pubAK\ (priAK\ n'),$
*Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK* $n'$)))$|\!\} \in synth\ (analz\ ?A)$
**by** (*rule synth-analz-snd*)
**hence** *Auth-Data* (*priAK* $n'$) $b \in$ *synth* (*analz ?A*)
**by** (*rule synth-analz-fst*)
**hence** *Auth-Data* (*priAK* $n'$) $b \in$ *analz ?A* $\vee$
*Pri-AgrK* (*priAK* $n'$) $\in$ *analz ?A* $\wedge$ *Pri-AgrK* $b \in$ *analz ?A*
**by** (*rule synth-auth-data*)
**moreover {**
**assume** *Auth-Data* (*priAK* $n'$) $b \in$ *analz ?A*
**hence** *Auth-Data* (*priAK* $n'$) $b \in$ *parts ?A*
**by** (*rule subsetD* [*OF analz-parts-subset*])
**}**
**moreover {**
**assume** *Pri-AgrK* (*priAK* $n'$) $\in$ *analz ?A* $\wedge$ *Pri-AgrK* $b \in$ *analz ?A*
**hence** *Pri-AgrK* (*priAK* $n'$) $\in$ *analz ?A* **..**
**moreover have** *Pri-AgrK* (*priAK* $n'$) $\notin$ *analz ?A*
**using** *A* **by** (*rule pr-auth-key-analz*)
**ultimately have** *Auth-Data* (*priAK* $n'$) $b \in$ *parts ?A*
**by** *contradiction*

```
  }
  ultimately have Auth-Data (priAK n') b ∈ parts ?A ..
}
ultimately have Auth-Data (priAK n') b ∈ parts ?A ..
with A have ∃ n run. IntMapK (S (Card n, n, run)) = Some b
 by (rule pr-auth-data-parts)
then obtain n' and run' where IntMapK (S (Card n', n', run')) = Some b
 by blast
with A have Pri-AgrK b ∉ analz ?A
 by (rule pr-int-mapk-analz)
hence H: Pri-AgrK b ∉ A
 using A by (simp add: pr-pri-agrk-analz)
have ∃ X. Says n run 3 X (User m) (pubAK f) ∈ set evsFR4
 using A and D by (rule pr-ext-agrk-user-4)
then obtain X where Says n run 3 X (User m) (pubAK f) ∈ set evsFR4 ..
with A have
 (∃ 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 (∃ s a b d. ?P s a b d) ∨ ?Q)
 by (rule pr-ext-agrk-user-5)
moreover have I: ¬ ?Q
proof (rule notI, erule exE, erule conjE)
  fix b'
  assume ExtMapK (S (User m, n, run)) = Some b'
  hence b' = b
   using B by simp
  moreover assume Pri-AgrK b' ∈ A
  ultimately have Pri-AgrK b ∈ A
   by simp
  thus False
   using H by contradiction
qed
ultimately have ∃ s a b d. ?P s a b d
 by simp
then obtain s' and a' and b' and d where J: ?P s' a' b' d
 by blast
hence ExtAgrK (S (User m, n, run)) = Some (d * (s' + a' * b'))
 using D by simp
with A and C have
 {|Key (sesK (c * (d * (s' + a' * b')))), Agent (User m), Number n, Number run|}
∈ U
 by (rule pr-sesk-user-1)
moreover have K: Key (sesK (c * (d * (s' + a' * b')))) ∈ A
```

**using** $F$ **and** $J$ **by** *simp*
**ultimately have**
$(\exists\, d'\ e'.\ c * (d * (s' + a' * b')) = d' * e' \wedge$
*IntAgrK* $(S\ (Card\ n,\ n,\ run)) = Some\ d' \wedge$
*ExtAgrK* $(S\ (Card\ n,\ n,\ run)) = Some\ e') \vee$
$(\exists\, b.\ Pri\text{-}AgrK\ b \in A \wedge$
*ExtMapK* $(S\ (User\ m,\ n,\ run)) = Some\ b)$
(**is** $(\exists\, d\ e.\ ?P\ d\ e) \vee$ -)
**by** (*rule pr-sesk-user-3* [*OF A*])
**hence** $\exists\, d\ e.\ ?P\ d\ e$
**using** $I$ **by** *simp*
**then obtain** $d'$ **and** $e'$ **where** $L$: $?P\ d'\ e'$
**by** *blast*
**hence** $d' = d$
**using** $J$ **by** *simp*
**hence** $d * c * (s' + a' * b') = d * e'$
**using** $L$ **by** *simp*
**hence** $e' = c * (s' + a' * b')$
**using** $J$ **by** *simp*
**hence** $M$: $ExtAgrK\ (S\ (Card\ n,\ n,\ run)) = Some\ (c * (s' + a' * b'))$
**using** $L$ **by** *simp*
**have** $c * (d * (s' + a' * b')) = c * d * (s' + a' * b')$
**by** *simp*
**hence** $Key\ (sesK\ (c * (d * (s' + a' * b')))) = Key\ (sesK\ (c * d * (s' + a' * b')))$
**by** (*rule arg-cong*)
**hence** $Key\ (sesK\ (c * d * (s' + a' * b'))) \in A$
**using** $K$ **by** *simp*
**moreover have** $Key\ (sesK\ (c * d * (s' + a' * b'))) \notin A$
**proof** (*rule pr-ext-agrk-card* [*OF A*, *of n run*], *simp-all add*: $J\ M$)
**qed** (*rule pr-int-agrk-user-2* [*OF A*, *of m n run*], *simp-all add*: $C\ E$)
**ultimately show** *False*
**by** *contradiction*
**qed**

**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** (*simp add*: *pr-key-analz*, *erule protocol.induct*, *simp-all*,
$(rule\text{-}tac\ [1]\ impI)+$, $(rule\text{-}tac\ [2-3]\ impI)+$, $rule\text{-}tac\ [1-2]\ notI$, *simp-all*)
**fix** *evsR3 S A U s a b c d*
**assume**
$(evsR3,\ S,\ A,\ U) \in protocol$ **and**
$Says\ n\ run\ 5\ (User\ m)\ (Card\ n)$
$(Crypt\ (sesK\ (c * d * (s + a * b)))\ (Passwd\ m)) \in set\ evsR3$
**hence** $Key\ (sesK\ (c * d * (s + a * b))) \in U$
**by** (*rule pr-sesk-passwd*)

**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsFR3 S A U s a b c d*
**assume**
(*evsFR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*Says n run 5* (*User m*) (*Card n*)
(*Crypt* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) (*Passwd m*)) ∈ *set evsFR3*
**hence** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** (*rule pr-sesk-passwd*)
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsC5 S A U n run s a b c f g X* **and** *m* :: *nat*
**assume** (*evsC5*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *0* < *m*
**hence** *User m* ≠ *Spy*
**by** *simp*
**moreover assume**
*NonceS* (*S* (*User m*, *n*, *run*)) = *Some s* **and**
*IntMapK* (*S* (*User m*, *n*, *run*)) = *Some a* **and**
*ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b* **and**
*IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* **and**
*ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f* **and**
*Says n run 4 X* (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*))
{|*pubAK* (*c* ∗ (*s* + *a* ∗ *b*)), *Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}
∈ *set evsC5*
**ultimately show** *Key* (*sesK* (*c* ∗ *f*)) ∉ *A*
**by** (*rule pr-key-secrecy-aux*)
**qed**

**theorem** *pr-passwd-secrecy* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*User m* ≠ *Spy* ⟶
*Passwd m* ∉ *analz* (*A* ∪ *spies evs*)
**proof** (*erule protocol.induct*, *rule-tac* [!] *impI*, *simp-all add*: *analz-simp-insert-2*,
*rule contra-subsetD* [*OF analz-parts-subset*], *subst parts-simp*, *simp*, *blast*+,
*rule-tac* [*3*] *impI*)
**fix** *evsR1 S A U n s*
**assume**
*A*: *Passwd m* ∉ *analz* (*A* ∪ *spies evsR1*) **and**
*B*: (*evsR1*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *Pri-AgrK s* ∉ *U*
**show**
(*n* ∈ *bad* ⟶ *Passwd m* ∉ *analz* (*insert* (*Crypt* (*symK n*) (*Pri-AgrK s*))
(*insert* (*Pri-AgrK s*) (*A* ∪ *spies evsR1*)))) ∧

```
    (n ∉ bad ⟶ Passwd m ∉ analz (insert (Crypt (symK n) (Pri-AgrK s))
      (A ∪ spies evsR1)))
    (is (- ⟶ ?T) ∧ (- ⟶ ?T'))
  proof (rule conjI, rule-tac [!] impI)
    assume n ∈ bad
    hence Key (invK (symK n)) ∈ analz (A ∪ spies evsR1)
     using B by (simp add: pr-symk-analz invK-symK)
    hence Key (invK (symK n)) ∈ analz (insert (Pri-AgrK s) (A ∪ spies evsR1))
     by (rule rev-subsetD, rule-tac analz-mono, blast)
    moreover have analz (insert (Pri-AgrK s) (A ∪ spies evsR1)) =
      insert (Pri-AgrK s) (analz (A ∪ spies evsR1))
     using B and C by (rule pr-pri-agrk-unused)
    ultimately show ?T
     using A by (simp add: analz-crypt-in)
  next
    assume n ∉ bad
    hence Key (invK (symK n)) ∉ analz (A ∪ spies evsR1)
     using B by (simp add: pr-symk-analz invK-symK)
    thus ?T'
     using A by (simp add: analz-crypt-out)
  qed
next
  fix evsFR1 A m' s
  assume
    A: Passwd m ∉ analz (A ∪ spies evsFR1) and
    B: Crypt (symK m') (Pri-AgrK s) ∈ synth (analz (A ∪ spies evsFR1))
  thus Passwd m ∉ analz (insert (Crypt (symK m') (Pri-AgrK s)) (A ∪ spies
evsFR1))
  proof (cases Key (invK (symK m')) ∈ analz (A ∪ spies evsFR1),
   simp-all add: analz-crypt-in analz-crypt-out)
    case True
    have Crypt (symK m') (Pri-AgrK s) ∈ analz (A ∪ spies evsFR1) ∨
      Pri-AgrK s ∈ synth (analz (A ∪ spies evsFR1)) ∧
      Key (symK m') ∈ analz (A ∪ spies evsFR1)
      (is ?P ∨ ?Q)
     using B by (rule synth-crypt)
    moreover {
      assume ?P
      hence Pri-AgrK s ∈ analz (A ∪ spies evsFR1)
       using True by (rule analz.Decrypt)
    }
    moreover {
      assume ?Q
      hence Pri-AgrK s ∈ synth (analz (A ∪ spies evsFR1)) ..
      hence Pri-AgrK s ∈ analz (A ∪ spies evsFR1)
       by (rule synth-simp-intro, simp)
    }
    ultimately have Pri-AgrK s ∈ analz (A ∪ spies evsFR1) ..
    thus Passwd m ∉ analz (insert (Pri-AgrK s) (A ∪ spies evsFR1))
```

**using** $A$ **by** (*simp add*: *analz-simp-insert-1*)
**qed**
**next**
**fix** *evsC2 S A U a*
**assume**
*Passwd m* $\notin$ *analz* ($A \cup$ *spies evsC2*) **and**
(*evsC2*, *S*, *A*, *U*) $\in$ *protocol* **and**
*Pri-AgrK a* $\notin$ *U*
**thus** *Passwd m* $\notin$ *analz* (*insert* (*Pri-AgrK a*) ($A \cup$ *spies evsC2*))
**by** (*subst pr-pri-agrk-unused*, *simp-all*)
**next**
**fix** *evsC3 S A U c* **and** $m'$ :: *nat*
**assume**
*Passwd m* $\notin$ *analz* ($A \cup$ *spies evsC3*) **and**
(*evsC3*, *S*, *A*, *U*) $\in$ *protocol* **and**
*Pri-AgrK c* $\notin$ *U*
**thus** $m' = 0 \longrightarrow$ *Passwd m* $\notin$ *analz* (*insert* (*Pri-AgrK c*) ($A \cup$ *spies evsC3*))
**by** (*rule-tac impI*, *subst pr-pri-agrk-unused*, *simp-all*)
**next**
**fix** *evsR3 S A U s s' a b c d*
**assume** *Passwd m* $\notin$ *analz* ($A \cup$ *spies evsR3*)
**moreover assume**
(*evsR3*, *S*, *A*, *U*) $\in$ *protocol* **and**
*Key* (*sesK* ($c * d * (s + a * b)$)) $\notin$ *U*
(**is** *Key ?K* $\notin$ -)
**hence** *analz* (*insert* (*Key ?K*) ($A \cup$ *spies evsR3*)) =
*insert* (*Key ?K*) (*analz* ($A \cup$ *spies evsR3*))
**by** (*rule pr-key-unused*)
**ultimately show** $s' = s \wedge$ *Pri-AgrK c* $\in$ *analz* ($A \cup$ *spies evsR3*) $\longrightarrow$
*Passwd m* $\notin$ *analz* (*insert* (*Key ?K*) ($A \cup$ *spies evsR3*))
**by** *simp*
**next**
**fix** *evsFR3 S A U s a b c d*
**assume** *Passwd m* $\notin$ *analz* ($A \cup$ *spies evsFR3*)
**moreover assume**
(*evsFR3*, *S*, *A*, *U*) $\in$ *protocol* **and**
*Key* (*sesK* ($c * d * (s + a * b)$)) $\notin$ *U*
(**is** *Key ?K* $\notin$ -)
**hence** *analz* (*insert* (*Key ?K*) ($A \cup$ *spies evsFR3*)) =
*insert* (*Key ?K*) (*analz* ($A \cup$ *spies evsFR3*))
**by** (*rule pr-key-unused*)
**ultimately show** *Passwd m* $\notin$ *analz* (*insert* (*Key ?K*) ($A \cup$ *spies evsFR3*))
**by** *simp*
**next**
**fix** *evsC4 A c f*
**assume** *Passwd m* $\notin$ *analz* ($A \cup$ *spies evsC4*)
**thus** *Passwd m* $\notin$ *analz* (*insert* (*Crypt* (*sesK* ($c * f$)) (*pubAK f*)) ($A \cup$ *spies evsC4*))
**by** (*cases Key* (*invK* (*sesK* ($c * f$))) $\in$ *analz* ($A \cup$ *spies evsC4*),

*simp-all add*: *analz-crypt-in analz-crypt-out analz-simp-insert-2*)
**next**
**fix** *evsFC4 A s a b d e*
**assume** *Passwd m* ∉ *analz* (*A* ∪ *spies evsFC4*)
**thus** *Passwd m* ∉ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*)) (*pubAK* (*d* ∗ (*s* + *a* ∗ *b*))))
(*A* ∪ *spies evsFC4*))
**by** (*cases Key* (*invK* (*sesK* (*d* ∗ *e*))) ∈ *analz* (*A* ∪ *spies evsFC4*),
*simp-all add*: *analz-crypt-in analz-crypt-out analz-simp-insert-2*)
**next**
**fix** *evsR4 S A U n run b d e*
**let**
*?A* = *A* ∪ *spies evsR4* **and**
*?H* = *Hash* (*pubAK* (*priAK n*)) **and**
*?M* = {|*pubAK* (*priAK n*), *Crypt* (*priSK CA*) (*Hash* (*pubAK* (*priAK n*)))|}
**assume**
*A*: *Passwd m* ∉ *analz ?A* **and**
*B*: (*evsR4*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *IntMapK* (*S* (*Card n*, *n*, *run*)) = *Some b*
**show** *Passwd m* ∉ *analz* (*insert* (*Crypt* (*sesK* (*d* ∗ *e*))
{|*pubAK e*, *Auth-Data* (*priAK n*) *b*, *?M*|}) *?A*)
**proof** (*cases Key* (*invK* (*sesK* (*d* ∗ *e*))) ∈ *analz ?A*,
*simp-all add*: *analz-crypt-in analz-crypt-out analz-mpair analz-simp-insert-2 A*)
**have** *Key* (*pubSK CA*) ∈ *analz ?A*
**using** *B* **by** (*rule pr-valid-key-analz*)
**hence** *D*: *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?A*) =
{*Crypt* (*priSK CA*) *?H*, *?H*} ∪ *analz ?A*
**by** (*simp add*: *analz-crypt-in analz-simp-insert-2*)
**have** *Pri-AgrK* (*priAK n*) ∉ *analz ?A*
**using** *B* **by** (*rule pr-auth-key-analz*)
**hence** *E*: *Pri-AgrK* (*priAK n*) ∉ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?A*)
**using** *D* **by** *simp*
**have** *Pri-AgrK b* ∉ *analz ?A*
**using** *B* **and** *C* **by** (*rule pr-int-mapk-analz*)
**hence** *F*: *Pri-AgrK b* ∉ *analz* (*insert* (*Crypt* (*priSK CA*) *?H*) *?A*)
**using** *D* **by** *simp*
**show** *Passwd m* ∉ *analz* (*insert* (*Auth-Data* (*priAK n*) *b*) (*insert ?M ?A*))
**proof** ((*subst insert-commute*, *simp add*: *analz-mpair analz-simp-insert-2*)+,
*subst analz-auth-data-out* [*OF E F*])
**qed** (*simp add*: *A D*)
**qed**
**next**
**fix** *evsFR4 S A U s a b c f g*
**let**
*?A* = *A* ∪ *spies evsFR4* **and**
*?H* = *Hash* (*pubAK g*) **and**
*?M* = {|*pubAK g*, *Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|} **and**
*?M′* = {|*pubAK* (*c* ∗ (*s* + *a* ∗ *b*)), *Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}

**assume**
*A*: *Passwd m* ∉ *analz ?A* **and**
*B*: (*evsFR4*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *Crypt* (*sesK* (*c* ∗ *f*)) *?M′* ∈ *synth* (*analz ?A*)
**have** *D*:
*Key* (*invK* (*sesK* (*c* ∗ *f*))) ∈ *analz ?A* ⟶
*Pri-AgrK b* ∈ *analz ?A* ∨ *Pri-AgrK g* ∈ *analz ?A* ⟶
*Pri-AgrK b* ∈ *analz ?A* ∧ *Pri-AgrK g* ∈ *analz ?A*
(**is** *?P* ⟶ *?Q* ⟶ *?R*)
**proof** (*rule impI*)+
**assume** *?P* **and** *?Q*
**have** *Crypt* (*sesK* (*c* ∗ *f*)) *?M′* ∈ *analz ?A* ∨
*?M′* ∈ *synth* (*analz ?A*) ∧ *Key* (*sesK* (*c* ∗ *f*)) ∈ *analz ?A*
**using** *C* **by** (*rule synth-crypt*)
**moreover** {
**assume** *Crypt* (*sesK* (*c* ∗ *f*)) *?M′* ∈ *analz ?A*
**hence** *?M′* ∈ *analz ?A*
**using** ‹*?P*› **by** (*rule analz.Decrypt*)
**hence** {|*Auth-Data g b*, *pubAK g*, *Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}
∈ *analz ?A*
**by** (*rule analz.Snd*)
**hence** *E*: *Auth-Data g b* ∈ *analz ?A*
**by** (*rule analz.Fst*)
**have** *?R*
**proof** (*rule disjE* [*OF* ‹*?Q*›])
**assume** *Pri-AgrK b* ∈ *analz ?A*
**moreover from** *this* **have** *Pri-AgrK g* ∈ *analz ?A*
**by** (*rule analz.Auth-Fst* [*OF E*])
**ultimately show** *?R* **..**
**next**
**assume** *Pri-AgrK g* ∈ *analz ?A*
**moreover from** *this* **have** *Pri-AgrK b* ∈ *analz ?A*
**by** (*rule analz.Auth-Snd* [*OF E*])
**ultimately show** *?R*
**by** *simp*
**qed**
}
**moreover** {
**assume** *?M′* ∈ *synth* (*analz ?A*) ∧
*Key* (*sesK* (*c* ∗ *f*)) ∈ *analz ?A*
**hence** *?M′* ∈ *synth* (*analz ?A*) **..**
**hence** {|*Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|} ∈ *synth* (*analz ?A*)
**by** (*rule synth-analz-snd*)
**hence** *Auth-Data g b* ∈ *synth* (*analz ?A*)
**by** (*rule synth-analz-fst*)
**hence** *Auth-Data g b* ∈ *analz ?A* ∨
*Pri-AgrK g* ∈ *analz ?A* ∧ *Pri-AgrK b* ∈ *analz ?A*
**by** (*rule synth-auth-data*)

```
    moreover {
      assume E: Auth-Data g b ∈ analz ?A
      have ?R
      proof (rule disjE [OF ‹?Q›])
        assume Pri-AgrK b ∈ analz ?A
        moreover from this have Pri-AgrK g ∈ analz ?A
         by (rule analz.Auth-Fst [OF E])
        ultimately show ?R ..
      next
        assume Pri-AgrK g ∈ analz ?A
        moreover from this have Pri-AgrK b ∈ analz ?A
         by (rule analz.Auth-Snd [OF E])
        ultimately show ?R
         by simp
      qed
    }
    moreover {
      assume Pri-AgrK g ∈ analz ?A ∧ Pri-AgrK b ∈ analz ?A
      hence ?R
       by simp
    }
    ultimately have ?R ..
  }
  ultimately show ?R ..
qed
show Passwd m ∉ analz (insert (Crypt (sesK (c * f)) ?M') ?A)
proof (cases Key (invK (sesK (c * f))) ∈ analz ?A,
 simp-all add: analz-crypt-in analz-crypt-out analz-mpair analz-simp-insert-2 A)
  assume E: Key (invK (sesK (c * f))) ∈ analz ?A
  have Key (pubSK CA) ∈ analz ?A
   using B by (rule pr-valid-key-analz)
  hence F: analz (insert (Crypt (priSK CA) ?H) ?A) =
    {Crypt (priSK CA) ?H, ?H} ∪ analz ?A
   by (simp add: analz-crypt-in analz-simp-insert-2)
  show Passwd m ∉ analz (insert (Auth-Data g b) (insert ?M ?A))
  proof (cases Pri-AgrK g ∈ analz ?A ∨ Pri-AgrK b ∈ analz ?A, simp-all)
    assume G: Pri-AgrK g ∈ analz ?A ∨ Pri-AgrK b ∈ analz ?A
    hence H: Pri-AgrK g ∈ analz (insert (Crypt (priSK CA) ?H) ?A) ∨
      Pri-AgrK b ∈ analz (insert (Crypt (priSK CA) ?H) ?A)
     using F by simp
    have I: Pri-AgrK b ∈ analz ?A ∧ Pri-AgrK g ∈ analz ?A
     using D and E and G by blast
    hence Pri-AgrK g ∈ analz (insert (Crypt (priSK CA) ?H) ?A)
     using F by simp
    hence J: Pri-AgrK g ∈ analz (insert (Pri-AgrK b)
      (insert (Crypt (priSK CA) ?H) ?A))
     by (rule rev-subsetD, rule-tac analz-mono, blast)
    have K: Pri-AgrK b ∈ analz (insert (Crypt (priSK CA) ?H) ?A)
     using F and I by simp
```

```
      show ?thesis
      proof ((subst insert-commute, simp add: analz-mpair analz-simp-insert-2)+,
       subst analz-auth-data-in [OF H], simp del: Un-insert-right,
       subst analz-simp-insert-1 [OF J], subst analz-simp-insert-1 [OF K])
      qed (simp add: A F)
    next
      assume G: Pri-AgrK g ∉ analz ?A ∧ Pri-AgrK b ∉ analz ?A
      hence H: Pri-AgrK g ∉ analz (insert (Crypt (priSK CA) ?H) ?A)
       using F by simp
      have I: Pri-AgrK b ∉ analz (insert (Crypt (priSK CA) ?H) ?A)
       using F and G by simp
      show ?thesis
      proof ((subst insert-commute, simp add: analz-mpair analz-simp-insert-2)+,
       subst analz-auth-data-out [OF H I])
      qed (simp add: A F)
    qed
  qed
next
  fix evsC5 S A U m' n run s a b c f g X
  let ?M = {|pubAK (c * (s + a * b)), Auth-Data g b, pubAK g,
    Crypt (priSK CA) (Hash (pubAK g))|}
  assume
    A: Passwd m ∉ analz (A ∪ spies evsC5) and
    B: 0 < m and
    C: (evsC5, S, A, U) ∈ protocol and
    D: NonceS (S (User m', n, run)) = Some s and
    E: IntMapK (S (User m', n, run)) = Some a and
    F: ExtMapK (S (User m', n, run)) = Some b and
    G: IntAgrK (S (User m', n, run)) = Some c and
    H: ExtAgrK (S (User m', n, run)) = Some f and
    I: Says n run 4 X (User m') (Crypt (sesK (c * f)) ?M) ∈ set evsC5
  from A show Passwd m ∉ analz (insert (Crypt (sesK (c * f)) (Passwd m'))
    (A ∪ spies evsC5))
  proof (cases Key (invK (sesK (c * f))) ∈ analz (A ∪ spies evsC5),
   simp-all add: analz-crypt-in analz-crypt-out analz-simp-insert-2, rule-tac notI)
    case True
    moreover assume m = m'
    hence User m' ≠ Spy
     using B by simp
    hence Key (sesK (c * f)) ∉ A
     by (rule pr-key-secrecy-aux [OF C - D E F G H I])
    hence Key (invK (sesK (c * f))) ∉ analz (A ∪ spies evsC5)
     using C by (simp add: pr-key-analz invK-sesK)
    ultimately show False
     by contradiction
  qed
next
  fix evsFC5 A n d e
  assume
```

```
    A: Passwd m ∉ analz (A ∪ spies evsFC5) and
    B: Crypt (sesK (d * e)) (Passwd n) ∈ synth (analz (A ∪ spies evsFC5))
  from A show Passwd m ∉ analz (insert (Crypt (sesK (d * e)) (Passwd n))
    (A ∪ spies evsFC5))
  proof (cases Key (invK (sesK (d * e))) ∈ analz (A ∪ spies evsFC5),
   simp-all add: analz-crypt-in analz-crypt-out analz-simp-insert-2, rule-tac notI)
    case True
    have Crypt (sesK (d * e)) (Passwd n) ∈ analz (A ∪ spies evsFC5) ∨
      Passwd n ∈ synth (analz (A ∪ spies evsFC5)) ∧
      Key (sesK (d * e)) ∈ analz (A ∪ spies evsFC5)
      (is ?P ∨ ?Q)
     using B by (rule synth-crypt)
    moreover {
      assume ?P
      hence Passwd n ∈ analz (A ∪ spies evsFC5)
       using True by (rule analz.Decrypt)
    }
    moreover {
      assume ?Q
      hence Passwd n ∈ synth (analz (A ∪ spies evsFC5)) ..
      hence Passwd n ∈ analz (A ∪ spies evsFC5)
       by (rule synth-simp-intro, simp)
    }
    ultimately have Passwd n ∈ analz (A ∪ spies evsFC5) ..
    moreover assume m = n
    hence Passwd n ∉ analz (A ∪ spies evsFC5)
     using A by simp
    ultimately show False
     by contradiction
  qed
next
  fix evsR5 A d e
  assume Passwd m ∉ analz (A ∪ spies evsR5)
  thus Passwd m ∉ analz (insert (Crypt (sesK (d * e)) (Number 0)) (A ∪ spies
evsR5))
   by (cases Key (invK (sesK (d * e))) ∈ analz (A ∪ spies evsR5),
    simp-all add: analz-crypt-in analz-crypt-out analz-simp-insert-2)
next
  fix evsFR5 A c f
  assume Passwd m ∉ analz (A ∪ spies evsFR5)
  thus Passwd m ∉ analz (insert (Crypt (sesK (c * f)) (Number 0)) (A ∪ spies
evsFR5))
   by (cases Key (invK (sesK (c * f))) ∈ analz (A ∪ spies evsFR5),
    simp-all add: analz-crypt-in analz-crypt-out analz-simp-insert-2)
qed
```

## 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*) ∈ *protocol* ⟹
*Crypt* (*sesK K*) (*Passwd m*) ∈ *parts* (*A* ∪ *spies evs*) ⟶
(∃ *n run*. *Says n run 5* (*User m*) (*Card n*) (*Crypt* (*sesK K*) (*Passwd m*)) ∈ *set evs*) ∨
(∃ *run*. *Says m run 5 Spy* (*Card m*) (*Crypt* (*sesK K*) (*Passwd m*)) ∈ *set evs*)
(**is** - ⟹ *?M* ∈ - ⟶ *?P evs* ∨ *?Q evs*)
**proof** (*erule protocol.induct*, *subst parts-simp*, (*simp*, *blast*)+)
**qed** (*simp-all add*: *parts-simp-insert parts-auth-data parts-crypt parts-mpair*, *blast*+)

**lemma** *pr-unique-run-1* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
{|*Key* (*sesK* (*d* ∗ *e*)), *Agent* (*User m*), *Number n′*, *Number run′*|} ∈ *U* ⟶
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* ⟶
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e* ⟶
*n′* = *n* ∧ *run′* = *run*
**proof** (*erule protocol.induct*, *simp-all*, *rule-tac* [*3*] *conjI*, (*rule-tac* [*1*−*4*] *impI*)+,
(*rule-tac* [*5*] *impI*)+, *simp-all*, (*erule-tac* [*2*−*3*] *conjE*)+, (*rule-tac* [!] *ccontr*))
**fix** *evsR3 S A U s′ a b c*
**assume** *c* ∗ (*s′* + *a* ∗ *b*) = *e*
**hence** *A*: *d* ∗ *e* = *c* ∗ *d* ∗ (*s′* + *a* ∗ *b*)
**by** *simp*
**assume**
(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
{|*Key* (*sesK* (*d* ∗ *e*)), *Agent* (*User m*), *Number n′*, *Number run′*|} ∈ *U*
**hence** *Key* (*sesK* (*d* ∗ *e*)) ∈ *U*
**by** (*rule pr-sesk-user-2*)
**hence** *Key* (*sesK* (*c* ∗ *d* ∗ (*s′* + *a* ∗ *b*))) ∈ *U*
**by** (*simp only*: *A*)
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s′* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsR3 S A U s a b c d′*
**assume** *Key* (*sesK* (*c* ∗ *d′* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**moreover assume** *sesK* (*d* ∗ *e*) = *sesK* (*c* ∗ *d′* ∗ (*s* + *a* ∗ *b*))
**with** *sesK-inj* **have** *d* ∗ *e* = *c* ∗ *d′* ∗ (*s* + *a* ∗ *b*)
**by** (*rule injD*)
**ultimately have** *Key* (*sesK* (*d* ∗ *e*)) ∉ *U*
**by** *simp*
**moreover assume**
(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**

*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**hence** *Key* (*sesK* (*d* ∗ *e*)) ∈ *U*
**by** (*rule pr-sesk-card*)
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsFR3 S A U s a b c d′*
**assume** *Key* (*sesK* (*c* ∗ *d′* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**moreover assume** *sesK* (*d* ∗ *e*) = *sesK* (*c* ∗ *d′* ∗ (*s* + *a* ∗ *b*))
**with** *sesK-inj* **have** *d* ∗ *e* = *c* ∗ *d′* ∗ (*s* + *a* ∗ *b*)
**by** (*rule injD*)
**ultimately have** *Key* (*sesK* (*d* ∗ *e*)) ∉ *U*
**by** *simp*
**moreover assume**
(*evsFR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**hence** *Key* (*sesK* (*d* ∗ *e*)) ∈ *U*
**by** (*rule pr-sesk-card*)
**ultimately show** *False*
**by** *contradiction*
**qed**

**lemma** *pr-unique-run-2*:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*IntAgrK* (*S* (*User m*, *n′*, *run′*)) = *Some c* ⟹
*ExtAgrK* (*S* (*User m*, *n′*, *run′*)) = *Some f* ⟹
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* ⟹
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e* ⟹
*d* ∗ *e* = *c* ∗ *f* ⟹
*n′* = *n* ∧ *run′* = *run*
**proof** (*frule pr-sesk-user-1*, *assumption*+, *drule sym* [*of d* ∗ *e*], *simp*)
**qed** (*rule pr-unique-run-1*)

**lemma** *pr-unique-run-3* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*IntAgrK* (*S* (*Card n′*, *n′*, *run′*)) = *Some d′* ⟶
*ExtAgrK* (*S* (*Card n′*, *n′*, *run′*)) = *Some e′* ⟶
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* ⟶
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e* ⟶
*d* ∗ *e* = *d′* ∗ *e′* ⟶
*n′* = *n* ∧ *run′* = *run*
**proof** (*erule protocol.induct*, *simp-all*, *rule-tac* [!] *conjI*, (*rule-tac* [!] *impI*)+,
*simp-all*, *rule-tac* [!] *ccontr*)
**fix** *evsR3 S A U n′ run′ s′ a b c*
**assume** *c* ∗ (*s′* + *a* ∗ *b*) = *e*
**hence** *A*: *d* ∗ *e* = *c* ∗ *d* ∗ (*s′* + *a* ∗ *b*)
**by** *simp*
**assume**

(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n'*, *n'*, *run'*)) = *Some d'* **and**
*ExtAgrK* (*S* (*Card n'*, *n'*, *run'*)) = *Some e'*
**hence** *Key* (*sesK* (*d'* ∗ *e'*)) ∈ *U*
**by** (*rule pr-sesk-card*)
**moreover assume** *d* ∗ *e* = *d'* ∗ *e'*
**ultimately have** *Key* (*sesK* (*c* ∗ *d* ∗ (*s'* + *a* ∗ *b*))) ∈ *U*
**using** *A* **by** *simp*
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s'* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**next**
**fix** *evsR3 S A U s' a b c*
**assume** *c* ∗ (*s'* + *a* ∗ *b*) = *e'*
**hence** *A*: *d'* ∗ *e'* = *c* ∗ *d'* ∗ (*s'* + *a* ∗ *b*)
**by** *simp*
**assume**
(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**hence** *Key* (*sesK* (*d* ∗ *e*)) ∈ *U*
**by** (*rule pr-sesk-card*)
**moreover assume** *d* ∗ *e* = *d'* ∗ *e'*
**ultimately have** *Key* (*sesK* (*c* ∗ *d'* ∗ (*s'* + *a* ∗ *b*))) ∈ *U*
**using** *A* **by** *simp*
**moreover assume** *Key* (*sesK* (*c* ∗ *d'* ∗ (*s'* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**qed**

**lemma** *pr-unique-run-4* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*Says n' run'* 5 *X* (*Card n'*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*)) ∈ *set evs* ⟶
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* ⟶
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e* ⟶
*n'* = *n* ∧ *run'* = *run*
**using** [[*simproc del*: *defined-all*]]
**proof** (*erule protocol.induct*, *simp-all*, (*rule-tac* [!] *impI*)+, *rule-tac* [*1*−*3*] *impI*,
*simp-all*, (*erule-tac* [*2*−*3*] *conjE*)+)
**fix** *evsR3 S A U n run s' a b c*
**assume** *c* ∗ (*s'* + *a* ∗ *b*) = *e*
**hence** *A*: *d* ∗ *e* = *c* ∗ *d* ∗ (*s'* + *a* ∗ *b*)
**by** *simp*
**assume**
(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*Says n' run'* 5 *X* (*Card n'*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*)) ∈ *set evsR3*
**hence** *Key* (*sesK* (*d* ∗ *e*)) ∈ *U*
**by** (*rule pr-sesk-passwd*)
**hence** *Key* (*sesK* (*c* ∗ *d* ∗ (*s'* + *a* ∗ *b*))) ∈ *U*

**by** (*simp only*: *A*)
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s*′ + *a* ∗ *b*))) ∉ *U*
**ultimately show** *n*′ = *n* ∧ *run*′ = *run*
**by** *contradiction*
**next**
**fix** *evsC5 S A U m n*′ *run*′ *c f*
**assume**
(*evsC5*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*User m*, *n*′, *run*′)) = *Some c* **and**
*ExtAgrK* (*S* (*User m*, *n*′, *run*′)) = *Some f* **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**moreover assume** *sesK* (*d* ∗ *e*) = *sesK* (*c* ∗ *f*)
**with** *sesK-inj* **have** *d* ∗ *e* = *c* ∗ *f*
**by** (*rule injD*)
**ultimately show** *n*′ = *n* ∧ *run*′ = *run*
**by** (*rule pr-unique-run-2*)
**next**
**fix** *evsFC5 S A U n*′ *run*′ *d*′ *e*′
**assume**
(*evsFC5*, *S*, *A*, *U*) ∈ *protocol* **and**
*IntAgrK* (*S* (*Card n*′, *n*′, *run*′)) = *Some d*′ **and**
*ExtAgrK* (*S* (*Card n*′, *n*′, *run*′)) = *Some e*′ **and**
*IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e*
**moreover assume** *sesK* (*d* ∗ *e*) = *sesK* (*d*′ ∗ *e*′)
**with** *sesK-inj* **have** *d* ∗ *e* = *d*′ ∗ *e*′
**by** (*rule injD*)
**ultimately show** *n*′ = *n* ∧ *run*′ = *run*
**by** (*rule pr-unique-run-3*)
**qed**

**theorem** *pr-user-authenticity* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*Says n run 5 X* (*Card n*) (*Crypt* (*sesK K*) (*Passwd m*)) ∈ *set evs* ⟶
*Says n run 5* (*User m*) (*Card n*) (*Crypt* (*sesK K*) (*Passwd m*)) ∈ *set evs*
**proof** (*erule protocol.induct*, *simp-all*, *rule impI*, *simp*)
**fix** *evsFC5 S A U n run d e*
**assume**
*A*: *Says n run 5 Spy* (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*)) ∈ *set evsFC5*
⟶
*Says n run 5* (*User n*) (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*)) ∈ *set evsFC5*
(**is** - ⟶ *?T*) **and**
*B*: (*evsFC5*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *IntAgrK* (*S* (*Card n*, *n*, *run*)) = *Some d* **and**
*D*: *ExtAgrK* (*S* (*Card n*, *n*, *run*)) = *Some e* **and**
*E*: *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *synth* (*analz* (*A* ∪ *spies evsFC5*))
**show** *n* = *0* ∨ *?T*

**proof** (*cases* $n = 0$, *simp-all*)
**assume** $0 < n$
**hence** *User* $n \neq$ *Spy*
**by** *simp*
**with** $B$ **have** $F$: *Passwd* $n \notin$ *analz* $(A \cup$ *spies evsFC5*$)$
**by** (*rule pr-passwd-secrecy*)
**have** *Crypt* (*sesK* $(d * e)$) (*Passwd* $n$) $\in$ *analz* $(A \cup$ *spies evsFC5*$) \vee$
*Passwd* $n \in$ *synth* (*analz* $(A \cup$ *spies evsFC5*$)) \wedge$
*Key* (*sesK* $(d * e)$) $\in$ *analz* $(A \cup$ *spies evsFC5*$)$
(**is** *?P* $\vee$ *?Q*)
**using** $E$ **by** (*rule synth-crypt*)
**moreover have** $\neg$ *?Q*
**proof**
**assume** *?Q*
**hence** *Passwd* $n \in$ *synth* (*analz* $(A \cup$ *spies evsFC5*$))$ **..**
**hence** *Passwd* $n \in$ *analz* $(A \cup$ *spies evsFC5*$)$
**by** (*rule synth-simp-intro*, *simp*)
**thus** *False*
**using** $F$ **by** *contradiction*
**qed**
**ultimately have** *?P*
**by** *simp*
**hence** *Crypt* (*sesK* $(d * e)$) (*Passwd* $n$) $\in$ *parts* $(A \cup$ *spies evsFC5*$)$
**by** (*rule subsetD* [*OF analz-parts-subset*])
**with** $B$ **have**
$(\exists\, n'\ run'.$ *Says* $n'$ *run'* 5 (*User* $n$) (*Card* $n'$) (*Crypt* (*sesK* $(d * e)$) (*Passwd* $n$))
$\in$ *set evsFC5*$) \vee$
$(\exists\, run'.$ *Says* $n$ *run'* 5 *Spy* (*Card* $n$) (*Crypt* (*sesK* $(d * e)$) (*Passwd* $n$))
$\in$ *set evsFC5*$)$
(**is** $(\exists\, n'\ run'.$ *?P* $n'$ *run'*$) \vee (\exists\, run'.$ *?Q run'*$)$)
**by** (*rule pr-passwd-parts*)
**moreover {**
**assume** $\exists\, n'\ run'.$ *?P* $n'$ *run'*
**then obtain** $n'$ **and** *run'* **where** *?P* $n'$ *run'*
**by** *blast*
**moreover from** *this* **have** $n' = n \wedge$ *run'* $=$ *run*
**by** (*rule pr-unique-run-4* [*OF B - C D*])
**ultimately have** *?T*
**by** *simp*
**}**
**moreover {**
**assume** $\exists\, run'.$ *?Q run'*
**then obtain** *run'* **where** *?Q run'* **..**
**moreover from** *this* **have** $n = n \wedge$ *run'* $=$ *run*
**by** (*rule pr-unique-run-4* [*OF B - C D*])
**ultimately have** *?Q run*
**by** *simp*
**with** $A$ **have** *?T* **..**

```
    }
    ultimately show ?T ..
  qed
qed

lemma pr-confirm-parts [rule-format]:
 (evs, S, A, U) ∈ protocol ⟹
    Crypt (sesK K) (Number 0) ∈ parts (A ∪ spies evs) ⟶
    Key (sesK K) ∉ A ⟶
  (∃ n run X.
    Says n run 5 X (Card n) (Crypt (sesK K) (Passwd n)) ∈ set evs ∧
    Says n run 5 (Card n) X (Crypt (sesK K) (Number 0)) ∈ set evs)
  (is - ⟹ - ⟶ - ⟶ ?P K evs)
using [[simproc del: defined-all]]
proof (erule protocol.induct, simp, subst parts-simp, simp, blast+,
 simp-all add: parts-simp-insert parts-auth-data parts-crypt parts-mpair,
 rule-tac [3] conjI, (rule-tac [!] impI)+, simp-all, blast+)
  fix evsFR5 S A U c f
  assume
    A: Crypt (sesK (c * f)) (Number 0) ∈ parts (A ∪ spies evsFR5) ⟶
      ?P (c * f) evsFR5 and
    B: (evsFR5, S, A, U) ∈ protocol and
    C: Key (sesK (c * f)) ∉ A and
    D: Crypt (sesK (c * f)) (Number 0) ∈ synth (analz (A ∪ spies evsFR5))
  show ?P (c * f) evsFR5
  proof −
    have Crypt (sesK (c * f)) (Number 0) ∈ analz (A ∪ spies evsFR5) ∨
      Number 0 ∈ synth (analz (A ∪ spies evsFR5)) ∧
      Key (sesK (c * f)) ∈ analz (A ∪ spies evsFR5)
     using D by (rule synth-crypt)
    moreover have Key (sesK (c * f)) ∉ analz (A ∪ spies evsFR5)
     using B and C by (simp add: pr-key-analz)
   ultimately have Crypt (sesK (c * f)) (Number 0) ∈ analz (A ∪ spies evsFR5)
     by simp
    hence Crypt (sesK (c * f)) (Number 0) ∈ parts (A ∪ spies evsFR5)
     by (rule subsetD [OF analz-parts-subset])
    with A show ?thesis ..
  qed
qed

lemma pr-confirm-says [rule-format]:
 (evs, S, A, U) ∈ protocol ⟹
    Says n run 5 X Spy (Crypt (sesK K) (Number 0)) ∈ set evs ⟶
  Says n run 5 Spy (Card n) (Crypt (sesK K) (Passwd n)) ∈ set evs
by (erule protocol.induct, simp-all)

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

```
  X = User m ∨ X = Spy
by (erule protocol.induct, simp-all)

lemma pr-unique-run-5 [rule-format]:
 (evs, S, A, U) ∈ protocol ⟹
    {|Key (sesK K), Agent (User m'), Number n', Number run'|} ∈ U ⟶
    {|Key (sesK K), Agent (User m), Number n, Number run|} ∈ U ⟶
  m = m' ∧ n = n' ∧ run = run'
using [[simproc del: defined-all]]
proof (erule protocol.induct, simp-all, blast, (rule conjI, rule impI)+,
 rule-tac [2] impI, (rule-tac [3] impI)+, rule-tac [4] conjI, (rule-tac [4-5] impI)+,
 simp-all, blast, rule-tac [!] ccontr)
  fix evsR3 S A U s a b c d
  assume
   (evsR3, S, A, U) ∈ protocol and
   {|Key (sesK (c * d * (s + a * b))), Agent (User m), Number n, Number run|} ∈
U
  hence Key (sesK (c * d * (s + a * b))) ∈ U
   by (rule pr-sesk-user-2)
  moreover assume Key (sesK (c * d * (s + a * b))) ∉ U
  ultimately show False
   by contradiction
next
  fix evsR3 S A U s a b c d
  assume
   (evsR3, S, A, U) ∈ protocol and
   {|Key (sesK (c * d * (s + a * b))), Agent (User m'), Number n', Number run'|}
∈ U
  hence Key (sesK (c * d * (s + a * b))) ∈ U
   by (rule pr-sesk-user-2)
  moreover assume Key (sesK (c * d * (s + a * b))) ∉ U
  ultimately show False
   by contradiction
next
  fix evsFR3 S A U s a b c d
  assume
   (evsFR3, S, A, U) ∈ protocol and
   {|Key (sesK (c * d * (s + a * b))), Agent (User m), Number n, Number run|} ∈
U
  hence Key (sesK (c * d * (s + a * b))) ∈ U
   by (rule pr-sesk-user-2)
  moreover assume Key (sesK (c * d * (s + a * b))) ∉ U
  ultimately show False
   by contradiction
next
  fix evsFR3 S A U s a b c d
  assume
   (evsFR3, S, A, U) ∈ protocol and
   {|Key (sesK (c * d * (s + a * b))), Agent (User m'), Number n', Number run'|}
```

$\in U$
**hence** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** (*rule pr-sesk-user-2*)
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *False*
**by** *contradiction*
**qed**

**lemma** *pr-unique-run-6*:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
{|*Key* (*sesK* (*c* ∗ *f*)), *Agent* (*User m*′), *Number n*′, *Number run*′|} ∈ *U* ⟹
*IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* ⟹
*ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f* ⟹
*m* = *m*′ ∧ *n* = *n*′ ∧ *run* = *run*′
**proof** (*frule pr-sesk-user-1*, *assumption*+)
**qed** (*rule pr-unique-run-5*)

**lemma** *pr-unique-run-7* [*rule-format*]:
(*evs*, *S*, *A*, *U*) ∈ *protocol* ⟹
*Says n*′ *run*′ *5* (*User m*′) (*Card n*′) (*Crypt* (*sesK K*) (*Passwd m*′)) ∈ *set evs*
⟶
{|*Key* (*sesK K*), *Agent* (*User m*), *Number n*, *Number run*|} ∈ *U* ⟶
*Key* (*sesK K*) ∉ *A* ⟶
*m*′ = *m* ∧ *n*′ = *n* ∧ *run*′ = *run*
**proof** (*erule protocol.induct*, *simp-all*, (*rule impI*)+, (*rule-tac* [*2*−*3*] *impI*)+,
(*erule-tac* [*3*] *conjE*)+, (*drule-tac* [*3*] *sym* [*of m*′])+, *drule-tac* [*3*] *sym* [*of 0*],
*simp-all*)
**fix** *evsR3 S A U n run s a b c d*
**assume**
(*evsR3*, *S*, *A*, *U*) ∈ *protocol* **and**
*Says n*′ *run*′ *5* (*User m*′) (*Card n*′)
(*Crypt* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) (*Passwd m*′)) ∈ *set evsR3*
**hence** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∈ *U*
**by** (*rule pr-sesk-passwd*)
**moreover assume** *Key* (*sesK* (*c* ∗ *d* ∗ (*s* + *a* ∗ *b*))) ∉ *U*
**ultimately show** *m*′ = *m* ∧ *n*′ = *n* ∧ *run*′ = *run*
**by** *contradiction*
**next**
**fix** *evsC5 S A U m*′ *n*′ *run*′ *c f*
**assume**
(*evsC5*, *S*, *A*, *U*) ∈ *protocol* **and**
{|*Key* (*sesK* (*c* ∗ *f*)), *Agent* (*User m*), *Number n*, *Number run*|} ∈ *U* **and**
*IntAgrK* (*S* (*User m*′, *n*′, *run*′)) = *Some c* **and**
*ExtAgrK* (*S* (*User m*′, *n*′, *run*′)) = *Some f*
**thus** *m*′ = *m* ∧ *n*′ = *n* ∧ *run*′ = *run*
**by** (*rule pr-unique-run-6*)
**next**
**fix** *evsFC5 S A U run*′ *d e*
**assume**

```
    A: Says 0 run' 5 Spy (Card 0) (Crypt (sesK (d * e)) (Passwd 0)) ∈ set evsFC5
⟶
      m = 0 ∧ n = 0 ∧ run' = run and
    B: (evsFC5, S, A, U) ∈ protocol and
    C: IntAgrK (S (Card 0, 0, run')) = Some d and
    D: ExtAgrK (S (Card 0, 0, run')) = Some e and
    E: Crypt (sesK (d * e)) (Passwd 0) ∈ synth (analz (A ∪ spies evsFC5)) and
    F: Key (sesK (d * e)) ∉ A
  have Crypt (sesK (d * e)) (Passwd 0) ∈ analz (A ∪ spies evsFC5) ∨
    Passwd 0 ∈ synth (analz (A ∪ spies evsFC5)) ∧
    Key (sesK (d * e)) ∈ analz (A ∪ spies evsFC5)
   using E by (rule synth-crypt)
  moreover have Key (sesK (d * e)) ∉ analz (A ∪ spies evsFC5)
   using B and F by (simp add: pr-key-analz)
  ultimately have Crypt (sesK (d * e)) (Passwd 0) ∈ analz (A ∪ spies evsFC5)
   by simp
  hence Crypt (sesK (d * e)) (Passwd 0) ∈ parts (A ∪ spies evsFC5)
   by (rule subsetD [OF analz-parts-subset])
  with B have
   (∃ n run. Says n run 5 Spy (Card n) (Crypt (sesK (d * e)) (Passwd 0))
      ∈ set evsFC5) ∨
    (∃ run. Says 0 run 5 Spy (Card 0) (Crypt (sesK (d * e)) (Passwd 0))
      ∈ set evsFC5)
    (is (∃ n run. ?P n run) ∨ (∃ run. ?Q run))
   by (rule pr-passwd-parts)
  moreover {
    assume ∃ n run. ?P n run
    then obtain n'' and run'' where ?P n'' run''
     by blast
    moreover from this have n'' = 0 ∧ run'' = run'
     by (rule pr-unique-run-4 [OF B - C D])
    ultimately have ?P 0 run'
     by simp
  }
  moreover {
    assume ∃ run. ?Q run
    then obtain run'' where ?Q run'' ..
    hence ∃ n. ?P n run'' ..
    then obtain n'' where ?P n'' run'' ..
    moreover from this have n'' = 0 ∧ run'' = run'
     by (rule pr-unique-run-4 [OF B - C D])
    ultimately have ?P 0 run'
     by simp
  }
  ultimately have ?P 0 run' ..
  with A show m = 0 ∧ n = 0 ∧ run' = run ..
qed

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** (*frule pr-sesk-user-1*, *assumption*+)
**qed** (*rule pr-unique-run-7*)

**lemma** *pr-unique-passwd-parts* [*rule-format*]:
$(evs,\ S,\ A,\ U) \in protocol \Longrightarrow$
$Crypt\ (sesK\ K)\ (Passwd\ m') \in parts\ (A \cup spies\ evs) \longrightarrow$
$Crypt\ (sesK\ K)\ (Passwd\ m) \in parts\ (A \cup spies\ evs) \longrightarrow$
$m' = m$
**using** [[*simproc del*: *defined-all*]]
**proof** (*erule protocol.induct*, *simp*, *subst parts-simp*, *simp*, *blast*+,
*simp-all add*: *parts-simp-insert parts-auth-data parts-crypt parts-mpair*,
*rule-tac* [!] *conjI*, (*rule-tac* [!] *impI*)+, *erule-tac* [!] *conjE*, *simp-all*)
**fix** *evsC5 S A U m'' n run s a b c f g X*
**assume**
*A*: $(evsC5,\ S,\ A,\ U) \in protocol$ **and**
*B*: $NonceS\ (S\ (User\ m'',\ n,\ run)) = Some\ s$ **and**
*C*: $IntMapK\ (S\ (User\ m'',\ n,\ run)) = Some\ a$ **and**
*D*: $ExtMapK\ (S\ (User\ m'',\ n,\ run)) = Some\ b$ **and**
*E*: $IntAgrK\ (S\ (User\ m'',\ n,\ run)) = Some\ c$ **and**
*F*: $ExtAgrK\ (S\ (User\ m'',\ n,\ run)) = Some\ f$ **and**
*G*: $Crypt\ (sesK\ (c * f))\ (Passwd\ m) \in parts\ (A \cup spies\ evsC5)$ **and**
*H*: $m' = m''$ **and**
*I*: $Says\ n\ run\ 4\ X\ (User\ m'')\ (Crypt\ (sesK\ (c * f))$
$\{|pubAK\ (c * (s + a * b)),\ Auth\text{-}Data\ g\ b,\ pubAK\ g,$
$Crypt\ (priSK\ CA)\ (Hash\ (pubAK\ g))|\} \in set\ evsC5$
**show** $m'' = m$
**proof** (*cases* $m'' = 0$, *rule classical*)
**case** *True*
**moreover assume** $m'' \neq m$
**ultimately have** *J*: $User\ m \neq Spy$
**using** *H* **by** *simp*
**have**
$(\exists\, n\ run.\ Says\ n\ run\ 5\ (User\ m)\ (Card\ n)\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))$
$\in set\ evsC5) \vee$
$(\exists\, run.\ Says\ m\ run\ 5\ Spy\ (Card\ m)\ (Crypt\ (sesK\ (c * f))\ (Passwd\ m))$
$\in set\ evsC5)$
$(\mathbf{is}\ (\exists\, n\ run.\ ?P\ n\ run) \vee (\exists\, run.\ ?Q\ run))$
**using** *A* **and** *G* **by** (*rule pr-passwd-parts*)
**moreover {**
**assume** $\exists\, n\ run.\ ?P\ n\ run$
**then obtain** $n'$ **and** $run'$ **where** *K*: $?P\ n'\ run'$
**by** *blast*

```
    with A and J have Key (sesK (c * f)) ∉ analz (A ∪ spies evsC5)
     by (rule pr-key-secrecy)
    hence Key (sesK (c * f)) ∉ A
     using A by (simp add: pr-key-analz)
    hence m = m'' ∧ n' = n ∧ run' = run
     by (rule pr-unique-run-8 [OF A K E F])
    hence ?thesis
     by simp
  }
  moreover {
    assume ∃ run. ?Q run
    then obtain run' where ?Q run' ..
    with A have K: ?P m run'
     by (rule pr-user-authenticity)
    with A and J have Key (sesK (c * f)) ∉ analz (A ∪ spies evsC5)
     by (rule pr-key-secrecy)
    hence Key (sesK (c * f)) ∉ A
     using A by (simp add: pr-key-analz)
    hence m = m'' ∧ m = n ∧ run' = run
     by (rule pr-unique-run-8 [OF A K E F])
    hence ?thesis
     by simp
  }
  ultimately show ?thesis ..
next
  case False
  hence User m'' ≠ Spy
   by simp
  hence J: Key (sesK (c * f)) ∉ A
   by (rule pr-key-secrecy-aux [OF A - B C D E F I])
  have
   (∃ n run. Says n run 5 (User m) (Card n) (Crypt (sesK (c * f)) (Passwd m))
      ∈ set evsC5) ∨
    (∃ run. Says m run 5 Spy (Card m) (Crypt (sesK (c * f)) (Passwd m))
      ∈ set evsC5)
    (is (∃ n run. ?P n run) ∨ (∃ run. ?Q run))
   using A and G by (rule pr-passwd-parts)
  moreover {
    assume ∃ n run. ?P n run
    then obtain n' and run' where ?P n' run'
     by blast
    hence m = m'' ∧ n' = n ∧ run' = run
     by (rule pr-unique-run-8 [OF A - E F J])
    hence ?thesis
     by simp
  }
  moreover {
    assume ∃ run. ?Q run
    then obtain run' where ?Q run' ..
```

```
      with A have ?P m run'
       by (rule pr-user-authenticity)
      hence m = m'' ∧ m = n ∧ run' = run
       by (rule pr-unique-run-8 [OF A - E F J])
      hence ?thesis
       by simp
    }
    ultimately show ?thesis ..
  qed
next
  fix evsC5 S A U m'' n run s a b c f g X
  assume
    A: (evsC5, S, A, U) ∈ protocol and
    B: NonceS (S (User m'', n, run)) = Some s and
    C: IntMapK (S (User m'', n, run)) = Some a and
    D: ExtMapK (S (User m'', n, run)) = Some b and
    E: IntAgrK (S (User m'', n, run)) = Some c and
    F: ExtAgrK (S (User m'', n, run)) = Some f and
    G: Crypt (sesK (c * f)) (Passwd m') ∈ parts (A ∪ spies evsC5) and
    H: m = m'' and
    I: Says n run 4 X (User m'') (Crypt (sesK (c * f))
      {|pubAK (c * (s + a * b)), Auth-Data g b, pubAK g,
       Crypt (priSK CA) (Hash (pubAK g))|}) ∈ set evsC5
  show m' = m''
  proof (cases m'' = 0, rule classical)
    case True
    moreover assume m' ≠ m''
    ultimately have J: User m' ≠ Spy
     using H by simp
    have
    (∃ n run. Says n run 5 (User m') (Card n) (Crypt (sesK (c * f)) (Passwd m'))
       ∈ set evsC5) ∨
     (∃ run. Says m' run 5 Spy (Card m') (Crypt (sesK (c * f)) (Passwd m'))
       ∈ set evsC5)
     (is (∃ n run. ?P n run) ∨ (∃ run. ?Q run))
     using A and G by (rule pr-passwd-parts)
    moreover {
      assume ∃ n run. ?P n run
      then obtain n' and run' where K: ?P n' run'
       by blast
      with A and J have Key (sesK (c * f)) ∉ analz (A ∪ spies evsC5)
       by (rule pr-key-secrecy)
      hence Key (sesK (c * f)) ∉ A
       using A by (simp add: pr-key-analz)
      hence m' = m'' ∧ n' = n ∧ run' = run
       by (rule pr-unique-run-8 [OF A K E F])
      hence ?thesis
       by simp
    }
```

```
  moreover {
    assume ∃ run. ?Q run
    then obtain run' where ?Q run' ..
    with A have K: ?P m' run'
     by (rule pr-user-authenticity)
    with A and J have Key (sesK (c * f)) ∉ analz (A ∪ spies evsC5)
     by (rule pr-key-secrecy)
    hence Key (sesK (c * f)) ∉ A
     using A by (simp add: pr-key-analz)
    hence m' = m'' ∧ m' = n ∧ run' = run
     by (rule pr-unique-run-8 [OF A K E F])
    hence ?thesis
     by simp
  }
  ultimately show ?thesis ..
next
  case False
  hence User m'' ≠ Spy
   by simp
  hence J: Key (sesK (c * f)) ∉ A
   by (rule pr-key-secrecy-aux [OF A - B C D E F I])
  have
  (∃ n run. Says n run 5 (User m') (Card n) (Crypt (sesK (c * f)) (Passwd m'))
     ∈ set evsC5) ∨
    (∃ run. Says m' run 5 Spy (Card m') (Crypt (sesK (c * f)) (Passwd m'))
     ∈ set evsC5)
    (is (∃ n run. ?P n run) ∨ (∃ run. ?Q run))
   using A and G by (rule pr-passwd-parts)
  moreover {
    assume ∃ n run. ?P n run
    then obtain n' and run' where ?P n' run'
     by blast
    hence m' = m'' ∧ n' = n ∧ run' = run
     by (rule pr-unique-run-8 [OF A - E F J])
    hence ?thesis
     by simp
  }
  moreover {
    assume ∃ run. ?Q run
    then obtain run' where ?Q run' ..
    with A have ?P m' run'
     by (rule pr-user-authenticity)
    hence m' = m'' ∧ m' = n ∧ run' = run
     by (rule pr-unique-run-8 [OF A - E F J])
    hence ?thesis
     by simp
  }
  ultimately show ?thesis ..
qed
```

**next**
**fix** *evsFC5 S A U n d e*
**assume**
*A*: *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *parts* (*A* ∪ *spies evsFC5*) ⟶
*n* = *m* **and**
*B*: (*evsFC5*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *synth* (*analz* (*A* ∪ *spies evsFC5*)) **and**
*D*: *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*) ∈ *parts* (*A* ∪ *spies evsFC5*)
**show** *n* = *m*
**proof** (*rule classical*)
**assume** *E*: *n* ≠ *m*
**have** *F*: *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *analz* (*A* ∪ *spies evsFC5*) ∨
*Passwd n* ∈ *synth* (*analz* (*A* ∪ *spies evsFC5*)) ∧
*Key* (*sesK* (*d* ∗ *e*)) ∈ *analz* (*A* ∪ *spies evsFC5*)
**using** *C* **by** (*rule synth-crypt*)
**have** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *analz* (*A* ∪ *spies evsFC5*)
**proof** (*rule disjE* [*OF F*], *assumption*, *erule conjE*, *cases n* = *0*)
**case** *True*
**hence** *G*: *User m* ≠ *Spy*
**using** *E* **by** *simp*
**have**
(∃ *n run. Says n run 5* (*User m*) (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*))
∈ *set evsFC5*) ∨
(∃ *run. Says m run 5 Spy* (*Card m*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*))
∈ *set evsFC5*)
(**is** (∃ *n run. ?P n run*) ∨ (∃ *run. ?Q run*))
**using** *B* **and** *D* **by** (*rule pr-passwd-parts*)
**moreover {**
**assume** ∃ *n run. ?P n run*
**then obtain** *n′* **and** *run* **where** *?P n′ run*
**by** *blast*
**with** *B* **and** *G* **have** *Key* (*sesK* (*d* ∗ *e*)) ∉ *analz* (*A* ∪ *spies evsFC5*)
**by** (*rule pr-key-secrecy*)
**}**
**moreover {**
**assume** ∃ *run. ?Q run*
**then obtain** *run* **where** *?Q run* **..**
**with** *B* **have** *?P m run*
**by** (*rule pr-user-authenticity*)
**with** *B* **and** *G* **have** *Key* (*sesK* (*d* ∗ *e*)) ∉ *analz* (*A* ∪ *spies evsFC5*)
**by** (*rule pr-key-secrecy*)
**}**
**ultimately have** *Key* (*sesK* (*d* ∗ *e*)) ∉ *analz* (*A* ∪ *spies evsFC5*) **..**
**moreover assume** *Key* (*sesK* (*d* ∗ *e*)) ∈ *analz* (*A* ∪ *spies evsFC5*)
**ultimately show** *?thesis*
**by** *contradiction*
**next**
**case** *False*

```
      hence User n ≠ Spy
       by simp
      with B have Passwd n ∉ analz (A ∪ spies evsFC5)
       by (rule pr-passwd-secrecy)
      moreover assume Passwd n ∈ synth (analz (A ∪ spies evsFC5))
      hence Passwd n ∈ analz (A ∪ spies evsFC5)
       by (rule synth-simp-intro, simp)
      ultimately show ?thesis
       by contradiction
    qed
    hence Crypt (sesK (d * e)) (Passwd n) ∈ parts (A ∪ spies evsFC5)
     by (rule subsetD [OF analz-parts-subset])
    with A show ?thesis ..
  qed
next
  fix evsFC5 S A U n d e
  assume
    A: Crypt (sesK (d * e)) (Passwd n) ∈ parts (A ∪ spies evsFC5) ⟶
      m' = n and
    B: (evsFC5, S, A, U) ∈ protocol and
    C: Crypt (sesK (d * e)) (Passwd n) ∈ synth (analz (A ∪ spies evsFC5)) and
    D: Crypt (sesK (d * e)) (Passwd m') ∈ parts (A ∪ spies evsFC5)
  show m' = n
  proof (rule classical)
    assume E: m' ≠ n
    have F: Crypt (sesK (d * e)) (Passwd n) ∈ analz (A ∪ spies evsFC5) ∨
      Passwd n ∈ synth (analz (A ∪ spies evsFC5)) ∧
      Key (sesK (d * e)) ∈ analz (A ∪ spies evsFC5)
     using C by (rule synth-crypt)
    have Crypt (sesK (d * e)) (Passwd n) ∈ analz (A ∪ spies evsFC5)
    proof (rule disjE [OF F], assumption, erule conjE, cases n = 0)
      case True
      hence G: User m' ≠ Spy
       using E by simp
      have
        (∃n run. Says n run 5 (User m') (Card n) (Crypt (sesK (d * e)) (Passwd
m'))
          ∈ set evsFC5) ∨
        (∃run. Says m' run 5 Spy (Card m') (Crypt (sesK (d * e)) (Passwd m'))
          ∈ set evsFC5)
        (is (∃n run. ?P n run) ∨ (∃run. ?Q run))
       using B and D by (rule pr-passwd-parts)
      moreover {
        assume ∃n run. ?P n run
        then obtain n' and run where ?P n' run
         by blast
        with B and G have Key (sesK (d * e)) ∉ analz (A ∪ spies evsFC5)
         by (rule pr-key-secrecy)
      }
```

```
    moreover {
     assume ∃ run. ?Q run
     then obtain run where ?Q run ..
     with B have ?P m' run
      by (rule pr-user-authenticity)
     with B and G have Key (sesK (d * e)) ∉ analz (A ∪ spies evsFC5)
      by (rule pr-key-secrecy)
    }
    ultimately have Key (sesK (d * e)) ∉ analz (A ∪ spies evsFC5) ..
    moreover assume Key (sesK (d * e)) ∈ analz (A ∪ spies evsFC5)
    ultimately show ?thesis
     by contradiction
   next
    case False
    hence User n ≠ Spy
     by simp
    with B have Passwd n ∉ analz (A ∪ spies evsFC5)
     by (rule pr-passwd-secrecy)
    moreover assume Passwd n ∈ synth (analz (A ∪ spies evsFC5))
    hence Passwd n ∈ analz (A ∪ spies evsFC5)
     by (rule synth-simp-intro, simp)
    ultimately show ?thesis
     by contradiction
   qed
   hence Crypt (sesK (d * e)) (Passwd n) ∈ parts (A ∪ spies evsFC5)
    by (rule subsetD [OF analz-parts-subset])
   with A show ?thesis ..
  qed
qed

theorem pr-card-authenticity [rule-format]:
 (evs, S, A, U) ∈ protocol ⟹
    Says n run 5 (User m) (Card n) (Crypt (sesK K) (Passwd m)) ∈ set evs ⟶
    Says n run 5 X (User m) (Crypt (sesK K) (Number 0)) ∈ set evs ⟶
  n = m ∧
  (Says m run 5 (Card m) (User m) (Crypt (sesK K) (Number 0)) ∈ set evs ∨
   Says m run 5 (Card m) Spy (Crypt (sesK K) (Number 0)) ∈ set evs)
proof (erule protocol.induct, simp-all, (rule-tac [1−2] impI)+, (erule-tac [2] conjE)+,
 (rule-tac [3] impI, rule-tac [3] conjI)+, rule-tac [4] disjI1, rule-tac [5] impI,
 (rule-tac [6] impI)+, simp-all)
  fix evsC5 S A U m n run s a b c f g X'
  assume
    A: Says n run 5 (User m) (Card n) (Crypt (sesK (c * f)) (Passwd m))
      ∈ set evsC5 ⟶
      n = m ∧
      (Says m run 5 (Card m) (User m) (Crypt (sesK (c * f)) (Number 0))
        ∈ set evsC5 ∨
       Says m run 5 (Card m) Spy (Crypt (sesK (c * f)) (Number 0))
        ∈ set evsC5) and
```

*B*: (*evsC5*, *S*, *A*, *U*) ∈ *protocol* **and**
*C*: *NonceS* (*S* (*User m*, *n*, *run*)) = *Some s* **and**
*D*: *IntMapK* (*S* (*User m*, *n*, *run*)) = *Some a* **and**
*E*: *ExtMapK* (*S* (*User m*, *n*, *run*)) = *Some b* **and**
*F*: *IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* **and**
*G*: *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f* **and**
*H*: *Says n run 4 X'* (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*))
{|*pubAK* (*c* ∗ (*s* + *a* ∗ *b*)), *Auth-Data g b*, *pubAK g*,
*Crypt* (*priSK CA*) (*Hash* (*pubAK g*))|}) ∈ *set evsC5* **and**
*I*: *Says n run 5 X* (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsC5*
**show** *n* = *m* ∧
(*Says m run 5* (*Card m*) (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsC5* ∨
*Says m run 5* (*Card m*) *Spy* (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsC5*)
**proof** (*cases m* = *0*)
**case** *True*
**hence** *Says n run 5 X Spy* (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsC5*
**using** *I* **by** *simp*
**with** *B* **have** *Says n run 5 Spy* (*Card n*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n*))
∈ *set evsC5*
**by** (*rule pr-confirm-says*)
**with** *B* **have** *J*: *Says n run 5* (*User n*) (*Card n*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n*))
∈ *set evsC5*
**by** (*rule pr-user-authenticity*)
**show** *?thesis*
**proof** (*cases n*)
**case** *0*
**hence** *Says n run 5* (*User m*) (*Card n*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*))
∈ *set evsC5*
**using** *J* **and** *True* **by** *simp*
**with** *A* **show** *?thesis* **..**
**next**
**case** *Suc*
**hence** *User n* ≠ *Spy*
**by** *simp*
**with** *B* **have** *Key* (*sesK* (*c* ∗ *f*)) ∉ *analz* (*A* ∪ *spies evsC5*)
**using** *J* **by** (*rule pr-key-secrecy*)
**hence** *Key* (*sesK* (*c* ∗ *f*)) ∉ *A*
**using** *B* **by** (*simp add*: *pr-key-analz*)
**hence** *n* = *m* ∧ *n* = *n* ∧ *run* = *run*
**by** (*rule pr-unique-run-8* [*OF B J F G*])
**hence** *Says n run 5* (*User m*) (*Card n*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*))
∈ *set evsC5*
**using** *J* **by** *simp*
**with** *A* **show** *?thesis* **..**
**qed**
**next**
**case** *False*

**have** *Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*) ∈ *spies evsC5*
**using** *I* **by** (*rule set-spies*)
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*) ∈ *A* ∪ *spies evsC5* **..**
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*) ∈ *parts* (*A* ∪ *spies evsC5*)
**by** (*rule parts.Inj*)
**moreover have** *User m* ≠ *Spy*
**using** *False* **by** *simp*
**hence** *J*: *Key* (*sesK* (*c* ∗ *f*)) ∉ *A*
**by** (*rule pr-key-secrecy-aux* [*OF B - C D E F G H*])
**ultimately have** ∃ *n run X*.
*Says n run 5 X* (*Card n*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n*)) ∈ *set evsC5* ∧
*Says n run 5* (*Card n*) *X* (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsC5*
**by** (*rule pr-confirm-parts* [*OF B*])
**then obtain** *n′* **and** *run′* **and** *X* **where**
*Says n′ run′ 5 X* (*Card n′*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n′*)) ∈ *set evsC5*
**by** *blast*
**with** *B* **have**
*Says n′ run′ 5* (*User n′*) (*Card n′*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n′*)) ∈ *set evsC5*
**by** (*rule pr-user-authenticity*)
**moreover from** *this* **have** *n′* = *m* ∧ *n′* = *n* ∧ *run′* = *run*
**by** (*rule pr-unique-run-8* [*OF B - F G J*])
**ultimately have**
*Says n run 5* (*User m*) (*Card n*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*)) ∈ *set evsC5*
**by** *auto*
**with** *A* **show** *?thesis* **..**
**qed**
**next**
**fix** *evsFC5 S A U run d e*
**assume**
*Says 0 run 5 Spy* (*Card 0*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd 0*)) ∈ *set evsFC5* ⟶
*Says 0 run 5* (*Card 0*) *Spy* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) ∈ *set evsFC5*
**moreover assume**
(*evsFC5*, *S*, *A*, *U*) ∈ *protocol* **and**
*Says 0 run 5 X Spy* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) ∈ *set evsFC5*
**hence** *Says 0 run 5 Spy* (*Card 0*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd 0*)) ∈ *set evsFC5*
**by** (*rule pr-confirm-says*)
**ultimately show**
*Says 0 run 5* (*Card 0*) *Spy* (*Crypt* (*sesK* (*d* ∗ *e*)) (*Number 0*)) ∈ *set evsFC5* **..**
**next**
**fix** *evsR5 S A U n run d e X*
**assume** (*evsR5*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *Says n run 5 X* (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*))
∈ *set evsR5*
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *spies evsR5*
**by** (*rule set-spies*)
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *A* ∪ *spies evsR5* **..**

**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *parts* (*A* ∪ *spies evsR5*)
**by** (*rule parts.Inj*)
**moreover assume** *Says n run 5 X* (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*))
∈ *set evsR5*
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*) ∈ *spies evsR5*
**by** (*rule set-spies*)
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*) ∈ *A* ∪ *spies evsR5* **..**
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*) ∈ *parts* (*A* ∪ *spies evsR5*)
**by** (*rule parts.Inj*)
**ultimately show** *n* = *m*
**by** (*rule pr-unique-passwd-parts*)
**next**
**fix** *evsR5 S A U n run d e X*
**assume** (*evsR5*, *S*, *A*, *U*) ∈ *protocol*
**moreover assume** *Says n run 5 X* (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*))
∈ *set evsR5*
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*) ∈ *spies evsR5*
**by** (*rule set-spies*)
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*) ∈ *A* ∪ *spies evsR5* **..**
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd m*) ∈ *parts* (*A* ∪ *spies evsR5*)
**by** (*rule parts.Inj*)
**moreover assume** *Says n run 5 X* (*Card n*) (*Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*))
∈ *set evsR5*
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *spies evsR5*
**by** (*rule set-spies*)
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *A* ∪ *spies evsR5* **..**
**hence** *Crypt* (*sesK* (*d* ∗ *e*)) (*Passwd n*) ∈ *parts* (*A* ∪ *spies evsR5*)
**by** (*rule parts.Inj*)
**ultimately show** *m* = *n*
**by** (*rule pr-unique-passwd-parts*)
**next**
**fix** *evsR5 n′ run′ d e X*
**assume** *n* = *m* ∧
(*Says m run 5* (*Card m*) (*User m*) (*Crypt* (*sesK K*) (*Number 0*)) ∈ *set evsR5*
∨
*Says m run 5* (*Card m*) *Spy* (*Crypt* (*sesK K*) (*Number 0*)) ∈ *set evsR5*)
**thus**
*m* = *n′* ∧ *run* = *run′* ∧ *m* = *n′* ∧ *User m* = *X* ∧ *sesK K* = *sesK* (*d* ∗ *e*) ∨
*Says m run 5* (*Card m*) (*User m*) (*Crypt* (*sesK K*) (*Number 0*)) ∈ *set evsR5* ∨
*m* = *n′* ∧ *run* = *run′* ∧ *m* = *n′* ∧ *Spy* = *X* ∧ *sesK K* = *sesK* (*d* ∗ *e*) ∨
*Says m run 5* (*Card m*) *Spy* (*Crypt* (*sesK K*) (*Number 0*)) ∈ *set evsR5*
**by** *blast*
**next**
**fix** *evsFR5 S A U m n run c f*
**assume**
*A*: (*evsFR5*, *S*, *A*, *U*) ∈ *protocol* **and**
*B*: *0* < *m* **and**
*C*: *IntAgrK* (*S* (*User m*, *n*, *run*)) = *Some c* **and**
*D*: *ExtAgrK* (*S* (*User m*, *n*, *run*)) = *Some f* **and**

*E*: *Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*) ∈ *synth* (*analz* (*A* ∪ *spies evsFR5*)) **and**
*F*: *Says n run 5* (*User m*) (*Card n*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*)) ∈ *set evsFR5*
**have** *User m* ≠ *Spy*
**using** *B* **by** *simp*
**with** *A* **have** *G*: *Key* (*sesK* (*c* ∗ *f*)) ∉ *analz* (*A* ∪ *spies evsFR5*)
**using** *F* **by** (*rule pr-key-secrecy*)
**moreover have** *Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*) ∈ *analz* (*A* ∪ *spies evsFR5*) ∨
*Number 0* ∈ *synth* (*analz* (*A* ∪ *spies evsFR5*)) ∧
*Key* (*sesK* (*c* ∗ *f*)) ∈ *analz* (*A* ∪ *spies evsFR5*)
**using** *E* **by** (*rule synth-crypt*)
**ultimately have** *Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*) ∈ *analz* (*A* ∪ *spies evsFR5*)
**by** *simp*
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*) ∈ *parts* (*A* ∪ *spies evsFR5*)
**by** (*rule subsetD* [*OF analz-parts-subset*])
**moreover have** *H*: *Key* (*sesK* (*c* ∗ *f*)) ∉ *A*
**using** *A* **and** *G* **by** (*simp add*: *pr-key-analz*)
**ultimately have** ∃ *n run X*.
*Says n run 5 X* (*Card n*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n*)) ∈ *set evsFR5* ∧
*Says n run 5* (*Card n*) *X* (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsFR5*
**by** (*rule pr-confirm-parts* [*OF A*])
**then obtain** *n′* **and** *run′* **and** *X* **where** *I*:
*Says n′ run′ 5 X* (*Card n′*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n′*)) ∈ *set evsFR5* ∧
*Says n′ run′ 5* (*Card n′*) *X* (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsFR5*
**by** *blast*
**hence**
*Says n′ run′ 5 X* (*Card n′*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n′*)) ∈ *set evsFR5* **..**
**with** *A* **have** *J*:
*Says n′ run′ 5* (*User n′*) (*Card n′*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n′*))
∈ *set evsFR5*
**by** (*rule pr-user-authenticity*)
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n′*) ∈ *spies evsFR5*
**by** (*rule set-spies*)
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n′*) ∈ *A* ∪ *spies evsFR5* **..**
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd n′*) ∈ *parts* (*A* ∪ *spies evsFR5*)
**by** (*rule parts.Inj*)
**moreover have** *Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*) ∈ *spies evsFR5*
**using** *F* **by** (*rule set-spies*)
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*) ∈ *A* ∪ *spies evsFR5* **..**
**hence** *Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*) ∈ *parts* (*A* ∪ *spies evsFR5*)
**by** (*rule parts.Inj*)
**ultimately have** *n′* = *m*
**by** (*rule pr-unique-passwd-parts* [*OF A*])
**moreover from** *this* **have**
*Says m run′ 5* (*User m*) (*Card m*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*))
∈ *set evsFR5*
**using** *J* **by** *simp*
**hence** *m* = *m* ∧ *m* = *n* ∧ *run′* = *run*

**by** (*rule pr-unique-run-8* [*OF A - C D H*])
**hence** *K*: $n = m \wedge run' = run$
**by** *simp*
**ultimately have** *L*:
*Says m run 5 X* (*Card m*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*)) ∈ *set evsFR5* ∧
*Says m run 5* (*Card m*) *X* (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsFR5*
**using** *I* **by** *simp*
**moreover from** *this* **have**
*Says m run 5 X* (*Card m*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Passwd m*)) ∈ *set evsFR5* **..**
**with** *A* **have** $X = User\ m \vee X = Spy$
**by** (*rule pr-passwd-says*)
**thus** $n = m \wedge$
(*Says m run 5* (*Card m*) (*User m*) (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsFR5* ∨
*Says m run 5* (*Card m*) *Spy* (*Crypt* (*sesK* (*c* ∗ *f*)) (*Number 0*)) ∈ *set evsFR5*)
**by** (*rule disjE*, *insert L*, *simp-all add*: *K*)
**qed**

**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.