Specifying and Verifying Hysteresis Signature System with HOL-Z
ETH Technical Report 471

David Basin, ETH Zurich
Hironobu Kuruma, Hitachi Systems Development Laboratory
Kazuo Takaragi, Hitachi Systems Development Laboratory
Burkhart Wolff, ETH Zurich

January 12, 2005

Abstract

We report on a case-study in using the data-oriented modeling language Z to formalize a security architecture for administering digital signatures and its architectural security requirements. Within an embedding of Z in the higher-order logic Isabelle/HOL, we provide formal machine-checked proofs of the correctness of the architecture with respect to its requirements.

A formalization and verification of the same architecture has been previously carried out using the process-oriented modeling language PROMELA and the SPIN model checker. We use this as a basis for comparing these two different approaches to formalization (infinite state with rich data types versus finite state) and verification (theorem proving versus model checking).

Contents

1 Introduction

While there is increasing consensus about the usefulness of formal methods for developing and validating critical systems (e.g., safety, security, or mission critical), there are many options and schools of thought on how best to do this. Formal methods can be loosely characterized along different dimensions in terms of what views of the system they emphasize, the proof techniques used, etc. When most of the complexity of the system (be it software, hardware, protocols, etc.) stems from the way that processes interact, and the data manipulations are comparatively simple, then the use of a process-oriented modeling languages, like a process algebra or some kind of communicating automata, is typically favored and model checking is the preferred means of verification. Protocols are typically given as examples of such systems. On the other hand, when data is structured into rich data types (e.g., formalizing problem domains, interface requirements, and the like) that are subject to complex manipulations, then data-oriented modeling languages are considered superior and verification is carried out by theorem proving. But what about systems whose design encompasses both complex data and nontrivial interaction and whose requirements speak both of the operations on data and their temporal sequencing? Here there is little consensus and the options available range from using abstraction to simplify the data model to enable model checking, to theorem proving, to combined formal methods.

In this report, we look at an example of one such system: a security architecture used for a digital signature application. The architecture is based on the secure operating system DARMA (Hitachi's platform for Dependable Autonomous hard Realtime MAnagement), which is used to control the interaction between different subsystems, running on different operating platforms. In particular, DARMA is used to ensure data integrity by separating user API functions, which run on a potentially open system (e.g., connected to the Internet), from those that actually manipulate signature-relevant data, which run on a separate, protected system. Any model of this architecture must formalize both the processes that run on the different platforms and the data that the processes manipulate to produce signatures. Moreover, the modeling formalism must be capable of formalizing data-integrity requirements, formalized as temporal properties of how different data-stores can change.

In the previous case study [**?**], we described a process-oriented model that we created of this system. The model was based on a series of abstractions that mapped the system's infinite data domain to a small finite-state domain. The resulting model was then formalized as a collection of communicating processes, in PROMELA, the input language of the SPIN model checker [**?**]. Afterwards, the data integrity requirements were formalized as temporal properties and verified by model checking.

Here we provide a data-oriented model of the signature system. We describe the system's state and its state transitions in the specification language Z [**?**]. As Z is a very rich specification language, we also use Z to formalize, on top of the data model, a simple process model describing the system semantics in terms of the set of its traces. This provides a basis for formalizing the system's integrity requirements as trace requirements and carrying out verification by induction over the set of traces. In doing so, we show how the use of a sufficiently expressive data-modeling language also provides a foundation for formalizing a trace-based model of process interaction. Thus, there is no need to resort to different formal methods to formalize and combine the different system views. This can all be done within Z itself. Moreover, via the embedding of Z in higher-order logic (HOL-Z), we can prove system correctness by theorem proving within the Isabelle/HOL system. Perhaps surprisingly, we will show that in the hands of an experienced user, such specification and verification is not substantially more complex, and in some regards is considerably simpler, than working with the process-oriented view alone using a model checker. Moreover, much of the additional time is spent in formalizing and proving additional system invariants, which increases our understanding of, and confidence in, the correctness of the system.

The modeling approach that we take is closely related to that of [**?**], where a security architecture (for access control for a version control system) as well as the desired temporal security properties were formalized in Z. There, using the HOL-Z embedding, the architecture was validated with respect to its specification. Moreover, the embedding was also used to carry out data refinement.

Organization

In Section 2, we provide background material on HOL-Z. In Section 3, we provide an overview of the signature architecture, its security requirements, and our modeling approach. In Section 4, we present our specification of the architecture's data model. In Section 5, we describe our formalization of the architecture's behavior, its security properties, and the verification that the architecture has these properties. In Section 6, we compare with our previous study and draw conclusions.

2 Background on HOL-Z

For modeling and verification, we use the HOL-Z proof environment [**?**]. The logical basis of this environment is a structure-preserving, shallow embedding of the specification language Z within Isabelle/HOL [**?**]. Using this embedding, systems are specified directly using Z notation and idioms, and the specifications are formulas within higher-order logic. Isabelle/HOL is used to prove properties of these specifications, using derived rules tailored to the structure of Z specifications.

The HOL-Z proof environment is part of a tool suite that supports the construction of specifications in a "literate specification style". In particular, specifications can be constructed as LaTeX documents (such as this document) that contain Z syntax, typeset using standard Z macros, and macros for proving proof obligations. These are mixed together along with informal explanations. The suite employs the ZeTa system [**?**] to extract Z definitions from the LaTeX documents and to type check them prior to the translation to Isabelle/HOL. Afterwards, a script is generated where Isabelle is used to validate that all proof obligations are fulfilled. Note that if these obligations are to be discharged automatically, then proof scripts for constructing Isabelle proofs must also be embedded in the LaTeX document. That is the case for all proofs reported on in this document.[1]

In what follows, we present the signature architecture in such a literate specification style. In doing so, we include information of the form "**section *Prelude***", which marks the start of a formal document section (here Prelude), and "**section *Basics* parents *Prelude***", which marks the start of a section (here Basic) that depends on another section (here Prelude). Each such section corresponds to a distinct Isabelle theory, ordered in a theory hierarchy under the "parents" dependency. As Z specifications are intended to serve as formal documentation, we have kept our explanations to a minimum and assume that the reader is familiar with Z and its use. Background on these topics may be found, e.g., in [**?**, **?**, **?**].

3 The Signature Architecture

In this section, following [**?**], we provide an informal overview of the signature architecture.

[1] However, as Isabelle proof scripts are not particularly readable, they are not presented in the output of LaTeX and relegated to the appendix.

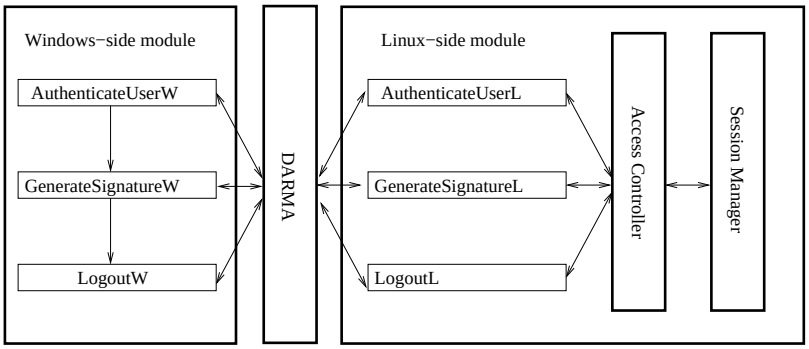

Figure 1: The signature architecture

3.1 Overview

The signature architecture is based on two ideas. The first is that of a *hysteresis signature* [**?**], which is a cryptographic approach designed to overcome the problem that for certain applications digital signatures should be valid for very long time periods. Hysteresis signatures address this problem by chaining signatures together in a way that the signature for each document signed depends on (hash values computed from) all previously signed documents. These chained signatures constitute a signature log and to forge even one signature in the log an attacker must forge (breaking the cryptographic functions behind) a chain of signatures.

The signature system must read the private keys of users from key stores, and read and update signature logs. Hence, the system's security relies on the confidentiality and integrity of this data. The second idea is to protect these using a secure operating platform. For this purpose, Hitachi's, DARMA system [**?**] is used to separate the user's operating system (in practice, Windows) from a second operating system used to manage system data (e.g., Linux). This technology plays a role analogous to network firewalls, but here the two systems are protected by controlling how functions in one system can call functions in the other. In this way, one can precisely limit how users access the functions and data for hysteresis signatures that reside in the Linux operating system space.

Our model is based on Hitachi documentation, which describes the signature architecture using diagrams (like Figures 1 and 2) and natural language text, as well as discussions with Hitachi engineers.

3.2 Functional Units and Dataflow

The signature architecture is organized into five modules, whose high-level structure is depicted in Figure 1, where the thick-lined boxes represent modules and the thin-lined boxes represent individual functions.

The first module contains three functions, which execute in the user operating system space. We call this the "Windows-side module" to reflect the (likely) scenario that they are part of an API available to programs running under the Windows operating system. These functions are essentially proxies. When called, they forward their parameters over the DARMA module to the corresponding functions in the second, protected, operating system, which is here called the "Linux-side module", again reflecting a likely implementation. There are two additional (sub)modules, each also executing on the second (Linux) operating system, which package data and functions for managing access control and sessions.

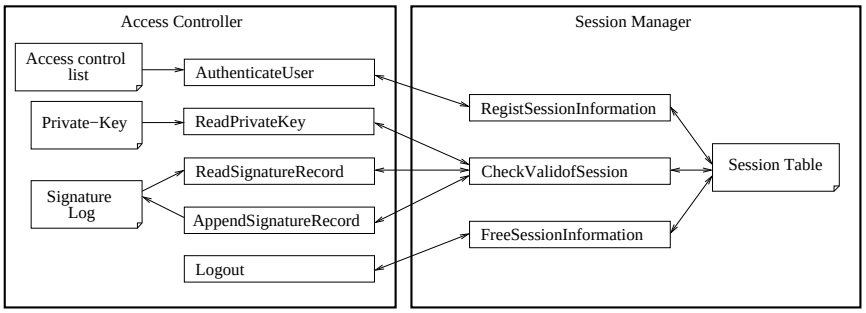

Figure 2: The Access Controller and Session Manger Modules

Parameters

Input:

username

Specifies name of user who generates hysteresis signature.

password

Specified *password* for *username*

Output:

SessionID

If user authentication is successful, $SessionID > 0$,
otherwise $SessionID \leq 0$.

Details

1. Sends *username*, *password* and *command* to Linux side using *CommunicateW*. The *command* is information from which the Linux-side module distinguishes the type of data receiving.
2. Outputs *SessionID* returned by *CommunicateW*.

Figure 3: Interface Description for *AuthenticateUserW*

To create a hysteresis signature, a user takes the following steps on the Windows side:

1. The user application calls *AuthenticateUserW* to authenticate the user and assign a session identifier.
2. The application calls *GenerateSignatureW* to generate a hysteresis signature.
3. The application calls *LogoutW* to logout, ending the session.

As explained above, each of these functions uses DARMA to call the corresponding function on the Linux side. DARMA restricts access from the Windows side to only these three functions. The Linux functions themselves may call any other Linux functions, including those of the *Access Controller*, which controls access to data (private keys, signature logs, and access control lists). The *Access Controller* in turn uses functions provided by the *Session Manager*, which manages session information (*SessionID*, etc.), as depicted in Figure 2.

The Hitachi documentation provides an interface description for each of these functions. Two representative examples are presented in Figures 3 and 4. These are the interface descriptions for the functions *AuthenticateUserL* and *AuthenticateUserW*. The former calls DARMA and returns a session identifier while the latter does the actual work of checking the password and communicating with the access controller.

Parameters

Input:

username: sent by *AuthenticateUserW* through *Darma*.

password: sent by *AuthenticateUserW* through *Darma*.

Output:

SessionID: If user authentication is successful, then $SessionID > 0$, otherwise $SessionID \leq 0$.

Details

1. Calculate hash value of *password* using the Keymate/Crypto API. If successful, go to step 2, otherwise set *SessionID* to *CrypotErr* ($\leq$0) and return.
2. Authenticate user using the function *AuthenticateUser* of *Access Controller*.
3. Output *SessionID* returned by *AuthenticateUser*.

Figure 4: Interface Description for *AuthenticateUserL*

3.3 Requirements

The Hitachi documentation also states three properties that the signature architecture should fulfill.

1. The signature architecture must authenticate a user before the user generates a hysteresis signature.
2. The signature architecture shall generate a hysteresis signature using the private key of an authenticated user.
3. The signature architecture must generate only one hysteresis signature per authentication.

3.4 Our Modeling Approach

Our formal model closely follows Hitachi's informal specification, described above. There is an identically named Z-schema for each of the three functions in both the Windows-side module (Section 4.6.1) and the Linux-side module (Section 4.6.2). These functions manipulate the session manager, access controller, and DARMA state (Section 4.4). The functions in the session manager and the access controller are formalized directly as axiomatic definitions (Section 4.3.2 and Section 4.3.3) respectively.

One key aspect of our model, which differs from the process model we reported on in [**?**], is the way we "wire" the modules and their functions together. In [**?**], we explicitly declared communication channels, which we used to model synchronous communication between the different parts. Here, we take an alternative approach, which is commonly used when subsystems are viewed as relations. Namely, we compose subsystems by conjunction, where communication ports are represented by shared variables, and we hide these ports by existentially quantifying them. In our model, we declare a schema DARMA (Section 4.3.2) that contains variables formalizing the different kinds of information (commands, user ids, hashed messages, etc.) communicated over DARMA. Each of the Windows-side and Linux-side schemas include a copy of this DARMA schema. When we wire together the different system operations (Section 4.7), we existentially quantify over the DARMA schema, which serves to internally wire together these ports, i.e., hide the communicated values from the outside.

Another key aspect is the way we formalize our security properties. The desired architecture properties are sets of traces that constitute temporal safety properties and thus a system is secure

when its traces are contained in (i.e., a subset of) the traces corresponding to each property. Hence, we formalize the set of system traces (Section 5.2). Afterwards, we formalize the system requirements directly as predicates over the traces. In our formalization, we talk and reason about explicit time points and relationships between time points. An alternative, of course, would be to directly formalize the operators of a logic like LTL over our traces via an embedding to partially encapsulate such pointed reasoning.

4 The Specification of the Signature Architecture

4.1 Extensions of the Z Environment

We start by extending Z with the usual type sum operator +, which represents the type whose elements are in the disjoint union of the two argument types, as well as the injections Inl and Inr. These operators are not part of the Z mathematical toolkit, but they are generally useful.

section $Prelude$

$[Prelude\_SUM]$

$$[X, Y]$$
$$\_ + \_ : X \times Y \to \mathbb{P}(Prelude\_SUM \times X \times Y)$$

$$[X, Y]$$
$$Inl : X \to (X + Y)$$
$$Inr : Y \to (X + Y)$$

4.2 Basic Types and Constants

section $Basics$ **parents** $Prelude$

We introduce the type $UNIT$ with just one element $unit$. Together we will use $UNIT$ and SUM to model error values.

$UNIT ::= unit$

Afterwards we introduce types relevant for the problem domain at hand, e.g., a type of valid user ids, user names, private keys, hash values, and session identifiers.

$[VALID\_USER\_ID, VALID\_SESSION\_ID, VALID\_PRI\_KEY]$
$[VALID\_SIGNATURE, USER\_NAME, CHAR]$

We then introduce types for different finite sets, which will contain names of commands, return values, and the like. Note that here we provide a vocabulary that goes beyond what is present in the informal description of Hitachi, for example, giving names to return values indicating error or denied access.

$$COMMAND ::= authenticate\_user \mid generate\_signature \mid logout \mid empty$$
$$PRI\_KEY\_READ\_ACCESS ::= accept\_read\_prikey \mid refuse\_read\_prikey$$
$$SIG\_LOG\_WRITE\_ACCESS ::= accept\_write\_siglog \mid refuse\_write\_siglog$$
$$ACCESS\_TYPE ::= read\_prikey \mid write\_siglog \mid read\_siglog$$
$$APPEND\_SIGNATURE\_RESULT ::= access\_denied \mid sig\_log\_updated$$
$$LOGOUT\_RESULT ::= invalid\_session\_id\_err \mid session\_terminated$$

The result of authentication, for example, is either a valid session identifier (a declared type) or a session error that identifies the reason for the authentication failure.

$$SESSION\_ERROR ::= crypt\_err \mid no\_user\_err \mid invalid\_pw\_err \mid same\_user\_err$$
$$SESSION\_ID == VALID\_SESSION\_ID + SESSION\_ERROR$$

Afterwards, to improve the specification's readability, we defined typed constants that inject the error elements into the $SESSION\_ID$ type.

$$\begin{array}{|l}
CRYPT_ERR : SESSION_ID \\
NO_USER_ERR : SESSION_ID \\
INVALID_PW_ERR : SESSION_ID \\
SAME_USER_ERR : SESSION_ID \\
AUTH_ERRORS : \mathbb{P}\, SESSION_ID \\
\hline
CRYPT_ERR = Inr\, crypt_err \\
NO_USER_ERR = Inr\, no_user_err \\
INVALID_PW_ERR = Inr\, invalid_pw_err \\
SAME_USER_ERR = Inr\, same_user_err \\
AUTH_ERRORS = \{CRYPT_ERR, NO_USER_ERR, \\
\qquad\qquad INVALID_PW_ERR, SAME_USER_ERR\}
\end{array}$$

We now introduce the types central to our specification. Along the way, we also introduce distinguished values that are either explicitly mentioned or implicitly assumed to exist in Hitachi's informal specification. For example, the signature generator may return the result $NULL$.

$$SIGNATURE == VALID\_SIGNATURE + UNIT$$
$$USER\_ID == VALID\_USER\_ID + UNIT$$
$$PRI\_KEY == VALID\_PRI\_KEY + UNIT$$

$$\begin{array}{|l}
NULL : SIGNATURE \\
NO_USER : USER_ID \\
NULL_KEY : PRI_KEY \\
\hline
NULL = Inr(unit) \\
NO_USER = Inr(unit) \\
NULL_KEY = Inr(unit)
\end{array}$$

The following definitions formalize the central data-stores manipulated by the system: the session table, access control list, private key table, and the signature log. For example, the access control list is a function from valid user identifiers to a user name and a hashed password. This data will later be used to define the system state in Section 4.4.

$$SESSION\_TABLE ==$$
$$(USER\_ID \setminus \{NO\_USER\}) \pfun$$
$$(SESSION\_ID \setminus AUTH\_ERRORS) \pfun$$
$$[pkra : PRI\_KEY\_READ\_ACCESS ;$$
$$slwa : SIG\_LOG\_WRITE\_ACCESS]$$

$$ACCESS\_CONTROL\_LIST ==$$
$$(USER\_ID \setminus \{NO\_USER\}) \pfun [usrn : USER\_NAME ; hpwd : \seq CHAR]$$
$$PRI\_KEY\_LIST == (USER\_ID \setminus \{NO\_USER\}) \pfun (PRI\_KEY \setminus \{NULL\_KEY\})$$
$$SIGNATURE\_LOG == (USER\_ID \setminus \{NO\_USER\}) \pfun (SIGNATURE \setminus \{NULL\})$$

4.3 Basic functions

In the following, we declare basic auxiliary functions. The function *new* returns a fresh $SESSION\_ID$ and *hash* returns a hash value of an arbitrary sequence of characters.

4.3.1 Primitives

$$new : \mathbb{F}\, SESSION\_ID \to (SESSION\_ID \setminus AUTH\_ERRORS)$$
$$\forall X : \mathbb{F}\, SESSION\_ID \bullet new(X) \notin X$$

$$hash : \seq CHAR \to \seq CHAR$$
$$hashFailure\_ : \mathbb{P}(\seq CHAR)$$

The function *hys_sig_gen* (for hysteresis signature generator) is the abstract interface to the KeyMate/Crypto API (a Hitachi library) for generating hysteresis signatures. It takes a hashed message (represented as a sequence of characters), a private key, and the previous signature and returns the generated signature.

$$hys\_sig\_gen :$$
$$\seq CHAR \times PRI\_KEY \times SIGNATURE \to SIGNATURE$$

4.3.2 Auxiliary Functions: Session Manager

section *SessionManager* **parents** *Basics*

The following are auxiliary definitions used to specify the session manager. For example, the first, for registering session information, returns a new session identifier for a user identifier that is not already in the session table, and otherwise it returns an error. These will be later used, in Section 4.6.2, to define how the session managers state is updated by Linux-side commands.

$$
\begin{array}{|l}
RegistSessionInformation : \\
\qquad USER\_ID \times SESSION\_TABLE \times \\
\qquad \mathbb{F}\, SESSION\_ID \\
\quad \rightarrow \\
\qquad SESSION\_ID \\
\hline
\forall uid : USER\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \; ; \\
\quad ssn\_IDs : \mathbb{F}\, SESSION\_ID \bullet \\
\quad RegistSessionInformation(uid, ssn\_tbl, ssn\_IDs) = \\
\quad \textbf{if}\; uid \in \mathsf{dom}\, ssn\_tbl \\
\quad \textbf{then}\; SAME\_USER\_ERR \\
\quad \textbf{else}\; new(ssn\_IDs)
\end{array}
$$

$$
\begin{array}{|l}
FreeSessionInformation : \\
\qquad SESSION\_ID \times SESSION\_TABLE \\
\quad \rightarrow \\
\qquad (LOGOUT\_RESULT \times SESSION\_TABLE) \\
\hline
\forall sid : SESSION\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \bullet \\
\quad FreeSessionInformation(sid, ssn\_tbl) = \\
\quad \textbf{if}\; sid \in \mathsf{dom}(\bigcup(\mathsf{ran}(ssn\_tbl))) \\
\quad \textbf{then}(session\_terminated, \\
\qquad \{uid : \mathsf{dom}\, ssn\_tbl | sid \notin \mathsf{dom}(ssn\_tbl(uid)) \bullet uid \mapsto ssn\_tbl(uid)\}) \\
\quad \textbf{else}(invalid\_session\_id\_err, ssn\_tbl)
\end{array}
$$

$$
\begin{array}{|l}
choose : \mathbb{P}(USER\_ID \setminus \{NO\_USER\}) \rightarrow USER\_ID \\
\hline
\forall X : \mathbb{P}(USER\_ID \setminus \{NO\_USER\}) \bullet \\
X \neq \{\} \Rightarrow choose(X) \in X
\end{array}
$$

The following function checks whether a session identifier is valid and, if so, returns the user identifier corresponding to the session identifier from the session table. The specification is fairly complex as (reflecting Hitachi's documentation) it must check whether the caller's access type is permitted by the session table and this may result in a new session table in the process.

$$
\begin{array}{l}
CheckValidofSession : \\
\qquad SESSION\_ID \times \\
\qquad ACCESS\_TYPE \times \\
\qquad SESSION\_TABLE \\
\quad \rightarrow \\
\qquad (USER\_ID \times SESSION\_TABLE) \\
\hline
\forall sid : SESSION\_ID ; acc\_typ : ACCESS\_TYPE ; \\
\quad ssn\_tbl : SESSION\_TABLE \bullet \\
\qquad CheckValidofSession(sid, acc\_typ, ssn\_tbl) = \\
\qquad \textbf{if } sid \in \mathsf{dom}(\bigcup(\mathsf{ran}(ssn\_tbl))) \\
\qquad \textbf{then if } acc\_typ = read\_prikey \\
\qquad \textbf{then if } accept\_read\_prikey \notin \\
\qquad\quad \{uid : \mathsf{dom}\ ssn\_tbl | sid \in \mathsf{dom}(ssn\_tbl(uid)) \bullet \\
\qquad\quad ((ssn\_tbl(uid))(sid)).pkra)\} \\
\qquad \textbf{then}(NO\_USER, ssn\_tbl) \\
\qquad \textbf{else}(\textbf{let } uid == choose\{uid : \mathsf{dom}\ ssn\_tbl | \\
\qquad\qquad sid \in \mathsf{dom}(ssn\_tbl(uid))\} \bullet \\
\qquad\quad (uid, ssn\_tbl \oplus \{uid \mapsto \{sid \mapsto \\
\qquad\qquad (\!| pkra == refuse\_read\_prikey, \\
\qquad\qquad slwa == ((ssn\_tbl(uid))(sid)).slwa) |\!\}\})) \\
\qquad \textbf{else if } acc\_typ = write\_siglog \\
\qquad \textbf{then if } accept\_write\_siglog \notin \\
\qquad\qquad \{uid : \mathsf{dom}\ ssn\_tbl | sid \in \mathsf{dom}(ssn\_tbl(uid)) \bullet \\
\qquad\qquad ((ssn\_tbl(uid))(sid)).slwa)\} \\
\qquad \textbf{then}(NO\_USER, ssn\_tbl) \\
\qquad \textbf{else}(\textbf{let } uid == choose\{uid : \mathsf{dom}\ ssn\_tbl | \\
\qquad\qquad sid \in \mathsf{dom}(ssn\_tbl(uid))\} \bullet \\
\qquad\quad (uid, ssn\_tbl \oplus \{uid \mapsto \{sid \mapsto \\
\qquad\quad (first(ssn\_tbl(uid))(sid), refuse\_write\_siglog))\}\})) \\
\qquad \textbf{else if } acc\_typ = read\_siglog \\
\qquad \textbf{then}(\textbf{let } uid == choose\{uid : \mathsf{dom}\ ssn\_tbl | \\
\qquad\qquad sid \in \mathsf{dom}(ssn\_tbl(uid))\} \bullet \\
\qquad\quad (uid, ssn\_tbl)) \\
\qquad \textbf{else}(NO\_USER, ssn\_tbl) \\
\qquad \textbf{else}(NO\_USER, ssn\_tbl)
\end{array}
$$

Note that Hitachi's informal specification uses an imperative style with implicit side-effects (ChackValidofSession modifies the underlying table according to the access-parameter). Moreover, possible exceptions may effect the subsequent control flow. For all auxiliary functions that compute values that may produce exceptions due to internal failures, we indicate this by introducing a variant $< X > Failure$. Due to the importance of the following failure predicate, we have renamed $CheckValidofSessionFailure$ (which would be the name according to previous conventions) to $isValidSession$.

$$
\begin{array}{l}
isValidSession\_ : \mathbb{P}(SESSION\_ID \times \\
\qquad\qquad ACCESS\_TYPE \times \\
\qquad\qquad SESSION\_TABLE) \\
\hline
\forall sid : SESSION\_ID \; ; \; acc\_typ : ACCESS\_TYPE \; ; \\
\quad ssn\_tbl : SESSION\_TABLE \bullet \\
\quad isValidSession(sid, acc\_typ, ssn\_tbl) \\
\iff \\
\quad first(CheckValidofSession(sid, acc\_typ, ssn\_tbl)) \neq NO\_USER
\end{array}
$$

4.3.3 Auxiliary functions: Access controller

section *AccessController* **parents** *SessionManager*

The specification of the access controller is based on a number of auxiliary declarations. Most of these are self explanatory and specify either how new authentication data is generated or how operations effect the session table.

$$
\begin{array}{l}
AuthenticateUser : \\
\qquad\qquad USER\_ID \times \mathrm{seq}\, CHAR \times \\
\qquad\qquad ACCESS\_CONTROL\_LIST \times \\
\qquad\qquad SESSION\_TABLE \times \mathbb{F}\, SESSION\_ID \\
\quad \rightarrow \\
\qquad\qquad SESSION\_ID \\
\hline
\forall uid : USER\_ID \; ; \; hpw : \mathrm{seq}\, CHAR \; ; \\
acc\_cnt\_lst : ACCESS\_CONTROL\_LIST \; ; \\
\quad ssn\_tbl : SESSION\_TABLE \; ; \\
\quad ssn\_IDs : \mathbb{F}\, SESSION\_ID \bullet \\
\quad AuthenticateUser(uid, hpw, acc\_cnt\_lst, ssn\_tbl, ssn\_IDs) = \\
\quad \textbf{if } uid \notin \mathsf{dom}\, acc\_cnt\_lst \\
\quad \textbf{then } NO\_USER\_ERR \\
\quad \textbf{else if } hpw \neq ((acc\_cnt\_lst(uid)).hpwd) \\
\quad \textbf{then } INVALID\_PW\_ERR \\
\quad \textbf{else } RegistSessionInformation(uid, ssn\_tbl, ssn\_IDs)
\end{array}
$$

$$
\begin{array}{l}
ReadPrivateKey : \\
\qquad\qquad SESSION\_ID \times \\
\qquad\qquad SESSION\_TABLE \times PRI\_KEY\_LIST \\
\quad \rightarrow \\
\qquad\qquad (PRI\_KEY \times SESSION\_TABLE) \\
\hline
\forall sid : SESSION\_ID \; ; \\
\quad ssn\_tbl : SESSION\_TABLE \; ; \\
\quad pri\_key\_lst : PRI\_KEY\_LIST \bullet \\
\quad ReadPrivateKey(sid, ssn\_tbl, pri\_key\_lst) = \\
\quad \textbf{if } \neg isValidSession(sid, read\_prikey, ssn\_tbl) \\
\quad \textbf{then} (NULL\_KEY, ssn\_tbl) \\
\quad \textbf{else let } R == CheckValidofSession(sid, read\_prikey, ssn\_tbl) \bullet \\
\qquad\qquad (pri\_key\_lst(firstR), secondR)
\end{array}
$$

$$
\begin{array}{|l}
ReadPrivateKeyFailure\_ : \mathbb{P}(SESSION\_ID \times \\
\qquad\qquad\qquad SESSION\_TABLE \times \\
\qquad\qquad\qquad PRI\_KEY\_LIST) \\
\hline
\forall sid : SESSION\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \; ; \\
\quad pri\_key\_lst : PRI\_KEY\_LIST \bullet \\
\quad ReadPrivateKeyFailure(sid, ssn\_tbl, pri\_key\_lst) \\
\iff \\
\quad first(ReadPrivateKey(sid, ssn\_tbl, pri\_key\_lst)) = NULL\_KEY
\end{array}
$$

$$
\begin{array}{|l}
ReadSignatureRecord : \\
\qquad SESSION\_ID \times \\
\qquad SESSION\_TABLE \times SIGNATURE\_LOG \\
\quad \rightarrow \\
\qquad SIGNATURE \\
\hline
\forall sid : SESSION\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \; ; \\
\quad sgn\_log : SIGNATURE\_LOG \bullet \\
ReadSignatureRecord(sid, ssn\_tbl, sgn\_log) = \\
\quad \textbf{if } \neg isValidSession(sid, read\_siglog, ssn\_tbl) \\
\quad \textbf{then } NULL \\
\quad \textbf{else } sgn\_log(first(CheckValidofSession(sid, read\_siglog, ssn\_tbl)))
\end{array}
$$

$$
\begin{array}{|l}
ReadSignatureRecordFailure\_ : \mathbb{P}(SESSION\_ID \times \\
\qquad\qquad\qquad SESSION\_TABLE \times PRI\_KEY\_LIST \times \\
\qquad\qquad\qquad SIGNATURE\_LOG) \\
\hline
\forall sid : SESSION\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \; ; \\
pri\_key\_lst : PRI\_KEY\_LIST \; ; \; sig\_log : SIGNATURE\_LOG \bullet \\
\quad ReadSignatureRecordFailure \\
\qquad (sid, ssn\_tbl, pri\_key\_lst, sig\_log) \\
\iff \\
\quad ReadSignatureRecord(sid, \\
\qquad second(ReadPrivateKey(sid, ssn\_tbl, pri\_key\_lst)), sig\_log) = \\
\quad NULL
\end{array}
$$

$$
\begin{array}{|l}
AppendSignatureRecord : \\
\qquad SESSION\_ID \times SESSION\_TABLE \times \\
\qquad SIGNATURE \times SIGNATURE\_LOG \\
\quad \rightarrow \\
\qquad (APPEND\_SIGNATURE\_RESULT \times \\
\qquad\quad (SIGNATURE\_LOG \times SESSION\_TABLE)) \\
\hline
\forall sid : SESSION\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \; ; \\
\quad sgn : SIGNATURE \; ; \; sgn\_log : SIGNATURE\_LOG \bullet \\
AppendSignatureRecord(sid, ssn\_tbl, sgn, sgn\_log) = \\
\quad \textbf{if } \neg isValidSession(sid, write\_siglog, ssn\_tbl) \\
\quad \textbf{then} (access\_denied, (sgn\_log, ssn\_tbl)) \\
\quad \textbf{else let } R == CheckValidofSession(sid, write\_siglog, ssn\_tbl) \bullet \\
\qquad (sig\_log\_updated, (sgn\_log \oplus \{first\, R \mapsto sgn\}, second\, R))
\end{array}
$$

$$
\begin{array}{l}
AppendSignatureRecordFailure\_ : \mathbb{P}(SESSION\_ID \\
\qquad \times SESSION\_TABLE \times PRI\_KEY\_LIST \\
\qquad \times SIGNATURE\_LOG \times \mathrm{seq}\, CHAR) \\
\hline
\forall sid : SESSION\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \; ; \\
\quad pri\_key\_lst : PRI\_KEY\_LIST \; ; \; sig\_log : SIGNATURE\_LOG \; ; \\
\quad hms : \mathrm{seq}\, CHAR \bullet \\
\quad AppendSignatureRecordFailure \\
\qquad (sid, ssn\_tbl, pri\_key\_lst, sig\_log, hms) \\
\iff \\
\quad first(AppendSignatureRecord(sid, ssn\_tbl, \\
\qquad SignatureGeneration(sid, ssn\_tbl, pri\_key\_lst, sig\_log, hms), sig\_log)) \\
\quad = access\_denied
\end{array}
$$

4.3.4 Auxiliary Functions: Hysteresis Signature

The function *SignatureGeneration* reads the private key, updates the *session_table* for the current session identifier *sid* to *read_access*, and reads in this context the previously stored signature. These three data items are then signed with the abstract crypto-primitive *hys_sig_gen*.

$$
\begin{array}{l}
SignatureGeneration : \\
\qquad SESSION\_ID \times SESSION\_TABLE \times \\
\qquad PRI\_KEY\_LIST \times SIGNATURE\_LOG \times \\
\qquad \mathrm{seq}\, CHAR \\
\quad \rightarrow \\
\qquad SIGNATURE \\
\hline
\forall sid : SESSION\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \; ; \\
\quad pri\_key\_lst : PRI\_KEY\_LIST \; ; \; sig\_log : SIGNATURE\_LOG \; ; \\
\quad hms : \mathrm{seq}\, CHAR \bullet \\
SignatureGeneration(sid, ssn\_tbl, pri\_key\_lst, sig\_log, hms) = \\
\quad hys\_sig\_gen(hms, \\
\qquad first(ReadPrivateKey(sid, ssn\_tbl, pri\_key\_lst)), \\
\qquad ReadSignatureRecord(sid, \\
\qquad\quad second(ReadPrivateKey(sid, ssn\_tbl, pri\_key\_lst)), \\
\qquad\qquad sig\_log))
\end{array}
$$

$$
\begin{array}{l}
SignatureGenerationFailure\_ : \mathbb{P}(SESSION\_ID \times SESSION\_TABLE \times \\
\qquad PRI\_KEY\_LIST \times SIGNATURE\_LOG \times \\
\qquad \mathrm{seq}\, CHAR) \\
\hline
\forall sid : SESSION\_ID \; ; \; ssn\_tbl : SESSION\_TABLE \; ; \\
\quad pri\_key\_lst : PRI\_KEY\_LIST \; ; \; sig\_log : SIGNATURE\_LOG \; ; \\
\quad hms : \mathrm{seq}\, CHAR \bullet \\
\quad SignatureGenerationFailure \\
\qquad (sid, ssn\_tbl, pri\_key\_lst, sig\_log, hms) \\
\iff \\
\quad SignatureGeneration(sid, ssn\_tbl, \\
\qquad pri\_key\_lst, sig\_log, hms) = NULL
\end{array}
$$

4.4 System Component States

section HSD **parents** $AccessController$

We are now ready to formalize the states of the various subsystems. The session manager maintains the session table and the set of session identifiers currently in use. The access controller maintains both the access control list and the corresponding list of private keys. The state of DARMA contains attributes for all the data items that DARMA must track during its operation. Finally, the hysteresis signature subsystem maintains the valid signature log.

SessionManager

$$session\_table : SESSION\_TABLE$$
$$session\_IDs : \mathbb{F}\, SESSION\_ID$$

$$\forall x, y : \mathsf{dom}\, session\_table \bullet$$
$$(\exists s : SESSION\_ID \bullet s \in \mathsf{dom}(session\_table(x)) \land s \in \mathsf{dom}(session\_table(y))) \Rightarrow x = y$$
$$\forall x : \mathsf{dom}\, session\_table \bullet \forall s : \mathsf{dom}(session\_table(x)) \bullet \mathsf{dom}(session\_table(x)) = \{s\}$$

The condition implies that session ids can be uniquely associated with their authenticated users.

AccessController

$$access\_control\_list : ACCESS\_CONTROL\_LIST$$
$$pri\_key\_list : PRI\_KEY\_LIST$$

$$\mathsf{dom}\, access\_control\_list = \mathsf{dom}\, pri\_key\_list$$
$$NULL\_KEY \notin \mathsf{ran}\, pri\_key\_list$$

DARMA

$$Command : COMMAND$$
$$User\_authentication\_uid : USER\_ID \setminus \{NO\_USER\}$$
$$User\_authentication\_pw : \mathrm{seq}\, CHAR$$
$$Signature\_generation\_sid : SESSION\_ID \setminus \{x : SESSION\_ERROR \bullet Inrx\}$$
$$Signature\_generation\_hmg : \mathrm{seq}\, CHAR$$
$$Logout\_ID : SESSION\_ID \setminus \{x : SESSION\_ERROR \bullet Inrx\}$$
$$Authentication : SESSION\_ID \setminus \{x : SESSION\_ERROR \bullet Inrx\}$$
$$Signature : SIGNATURE \setminus \{NULL\}$$
$$Result : LOGOUT\_RESULT$$

HysteresisSignature

$$signature\_log : SIGNATURE\_LOG$$

$$NULL \notin \mathsf{ran}\, signature\_log$$

4.5 Initial States

The initial states are formalized below. NoWe assume that the session manager and hysteresis signature states start with empty tables and logs. Note that the system specified does not change the state of the access controller and the security properties should hold for any such state. Hence no assumptions are made in this case.

SessionManagerInit

$SessionManager$

$session\_table = \{\}$
$session\_IDs = \{\}$

AccessControllerInit

$AccessController$

HysteresisSignatureInit

$HysteresisSignature$

$signature\_log = \{\}$

4.6 System Model Operations

4.6.1 Windows-Side Operations

The Windows operations represent the functions present as part of a Windows-side API that allows communication with DARMA. Unlike in our previous PROMELA model, we will *not* explicitly represent communication channels. Instead, as noted in Section 3.4, we model communication using shared values: here, the DARMA state. Later we will model wiring by existentially quantifying over this state, which, by hiding these shared variables, models internal communication.

Under this approach, it is straightforward to model the Windows-side operations. For example, AuthenticateUserW models $AuthenticateUserW$ given in Figure 3. Here the variables $User\_authenticate\_uid$, $User\_authenticate\_pw$, $Command$, and $Authentication$ are state variables from the DARMA state schema. The first two are set by the input values *userid*? and *password*?, $Command$ is the command named by the schema, and the output value *session_id*! is the $Authentication$ attribute of the DARMA schema.

The other two schemas, for generating signatures and logging out, are formalized similarly.

AuthenticateUserW

$userid? : USER\_ID$
$password? : \mathrm{seq}\, CHAR$
$session\_id! : SESSION\_ID$
$DARMA$

$User\_authentication\_uid = userid?$
$User\_authentication\_pw = password?$
$Command = authenticate\_user$
$session\_id! = Authentication$

GenerateSignatureW

$session\_id? : SESSION\_ID$
$message? : \mathrm{seq}\, CHAR$
$signature! : SIGNATURE$
$DARMA$

$Signature\_generation\_sid = session\_id?$
$Signature\_generation\_hmg = hash(message?)$
$Command = \textbf{if}\ hashFailure(message?)\ \textbf{then}\ empty$
$\quad\quad \textbf{else}\ generate\_signature$
$signature! = \textbf{if}\ hashFailure(message?)\ \textbf{then}\ NULL\ \textbf{else}\ Signature$

LogoutW

$session\_id? : SESSION\_ID$
$result! : LOGOUT\_RESULT$
$DARMA$

$Command = logout$
$Logout\_ID = session\_id?$
$result! = Result$

4.6.2 Linux-Side Operations

The Linux-side operations cause changes in the states of the session manager and the hysteresis signature. Note that as modeled, the access controller's state, which contains the access control list and the private key table (but excluding the signature log) never changes. Hitachi's specification (and hence, a fortiori, also ours) does not include a description of how this information can be altered by an administrator.

Linux-side user authentication is formalized by a single schema that models the informal interface description given in Figure 4. Step 1 of the informal description is reflected in the test of the hash value. Step 2 is modeled in the *else* branch, using the previously modeled function authenticate user. The last half of the specification specifies how to proceed in the case of successful ($Authentication \notin AUTH\_ERRORS$) and unsuccessful ($Authentication \in AUTH\_ERRORS$) authentication. The former specifies how the session manager's state (the session table and session identifiers) are updated. The latter specifies that these remain unchanged.

In these schemas, we follow the standard Z convention of using variables postfixed by "?" and "!" to denote inputs and outputs, respectively. These are used here to designate values coming from (and flowing back to) DARMA. Logically, these variables are determined by the DARMA state and could be eliminated. However, they help to maintain the correspondence between our formal specification and Hitachi's informal specification.

AuthenticateUserL

$\Delta SessionManager$
$\Xi HysteresisSignature$
$\Xi AccessController$
$username? : USER\_ID$
$password? : \mathrm{seq}\, CHAR$
$SessionID! : SESSION\_ID$
$DARMA$

$Command = authenticate\_user$
$Authentication = \textbf{if}\ hashFailure(User\_authentication\_pw)$
$\quad \textbf{then}\ CRYPT\_ERR$
$\quad \textbf{else}\ AuthenticateUser($
$\quad\quad User\_authentication\_uid,$
$\quad\quad hash(User\_authentication\_pw),$
$\quad\quad access\_control\_list,$
$\quad\quad session\_table,$
$\quad\quad session\_IDs)$
$session\_table' = \textbf{if}\ Authentication \notin AUTH\_ERRORS \quad \textbf{then}\ session\_table \cup$
$\quad \{User\_authentication\_uid \mapsto$
$\quad \{Authentication \mapsto$
$\quad \langle\!| pkra == accept\_read\_prikey,$
$\quad slwa == accept\_write\_siglog |\!\rangle\}\}$
$\quad \textbf{else}\ session\_table$
$session\_IDs' = \textbf{if}\ Authentication \notin AUTH\_ERRORS$
$\quad \textbf{then}\ session\_IDs \cup \{Authentication\} \quad \textbf{else}\ session\_IDs$
$username? = User\_authentication\_uid$
$password? = User\_authentication\_pw$
$SessionID! = Authentication$

To formalize *GenerateSignatureL*, we formalize the possible reasons for failure followed by how the signature is generated and the state is updated in case of success. Here our auxiliary functions are used to specify reading the private key, generating the signature, and updating (by appending) the signature record.

GenerateSignatureL

$\Delta SessionManager$
$\Delta HysteresisSignature$
$\Xi AccessController$
$SessionID? : SESSION\_ID$
$MsgHash? : \mathrm{seq}\, CHAR$
$signature! : SIGNATURE$
$DARMA$

$Command = generate\_signature$
$(Signature, (signature\_log', session\_table')) =$
if $ReadPrivateKeyFailure(Signature\_generation\_sid, session\_table, pri\_key\_list)$
$\vee ReadSignatureRecordFailure(Signature\_generation\_sid,$
$session\_table, pri\_key\_list, signature\_log)$
$\vee SignatureGenerationFailure(Signature\_generation\_sid,$
$session\_table, pri\_key\_list, signature\_log, Signature\_generation\_hmg)$
$\vee AppendSignatureRecordFailure(Signature\_generation\_sid,$
$session\_table, pri\_key\_list, signature\_log, Signature\_generation\_hmg)$
then $(NULL, (signature\_log, session\_table))$
else (**let** $prikey\_res == ReadPrivateKey(Signature\_generation\_sid,$
$session\_table, pri\_key\_list) \bullet$
let $sign == SignatureGeneration(Signature\_generation\_sid, session\_table,$
$pri\_key\_list, signature\_log, Signature\_generation\_hmg) \bullet$
let $app\_res == second(AppendSignatureRecord(Signature\_generation\_sid,$
$second(prikey\_res), sign, signature\_log)) \bullet$
$(sign, app\_res))$
$SessionID? = Signature\_generation\_sid$
$MsgHash? = Signature\_generation\_hmg$
$signature! = Signature$
$session\_IDs' = session\_IDs$

Finally, the Linux-side logout schema specifies a change to the session managers state whereby the tuple associated with $Logout\_ID$ is from the session table in the case of a successful ($session\_terminated$) log out.

LogoutL

$\Delta SessionManager$
$\Xi HysteresisSignature$
$\Xi AccessController$
$SessionID? : SESSION\_ID$
$result! : LOGOUT\_RESULT$
$DARMA$

$Command = logout$
$(Result, session\_table') =$
$FreeSessionInformation(Logout\_ID, session\_table)$
$SessionID? = Signature\_generation\_sid$
$session\_IDs' = session\_IDs$

The final schema formalizes the case of an improper command.

NopOperationL

$\Xi SessionManager$
$\Xi HysteresisSignature$
$\Xi AccessController$
$DARMA$

$Command \notin \{authenticate\_user, generate\_signature, logout\}$

4.7 Wiring Both Sides into the Architecture

section *HSDArch* **parents** *HSD*

We now "wire together" the architecture by specifying how the Windows-side operations interact with the Linux-side operations. The architectural pattern used here was explained in Section 3.4. Here each client-side operation is put in parallel (using conjunction) with the collection of server-side operations and the shared state of DARMA is used for communication and is hidden to represent internal communication.

$$ClientOperation == \\ AuthenticateUserW \lor GenerateSignatureW \lor LogoutW$$

$$ServerOperation == \\ AuthenticateUserL \lor GenerateSignatureL \lor LogoutL \lor NopOperationL$$

The client/server architecture is then built by putting the client and the server in parallel and by synchronizing them over DARMA:

$$System == \exists DARMA \bullet ClientOperation \land ServerOperation$$

The global state of the system is built by composing the states of the system components. The initial state is built analogously.

$$GlobalState == SessionManager \land HysteresisSignature \land AccessController$$
$$InitState == SessionManagerInit \land HysteresisSignatureInit \land AccessControllerInit$$

5 Security Properties

section *Analysis* **parents** *HSDArch*

5.1 General System Requirements

The hysteresis signature system specification explicitly formalizes both the system postulates, which constitute properties of the system environment, and the system requirements, which are the properties that should hold of the system.

Postulates

1. Any functions from the Windows side cannot access the Linux side except through the DARMA API.

2. User applications can only use the following functions: DARMA API, $AuthenticateUserW$, $LogoutW$, and $GenerateSignatureW$, and cannot use $CommunicateW$ and the functions from the Linux-side module.

3. The DARMA API cannot call any other functions of the Linux side except $CommunicateL$.

Requirements

1. The HSD (the Hysteresis Signature System using DARMA) shall authenticate a user before the user generates a hysteresis signature.

2. The HSD shall generate a hysteresis signature using the private key of an authenticated user.

3. The HSD shall generate only one hysteresis signature per authentication.

The requirements amount to temporal properties specified over possible system behaviors. To prove these we must first specify what these behaviors are. We do this by formalizing the set of system traces, which represents the different ways that the state of the composite system can evolve.

5.2 System Traces

The following formalizes the set of system traces, given the initial state and the global transitions of the system. Note that the transition relation ($Next$) allows interface functions to be called from the Windows side in any order (i.e., disjunction models nondeterministic choice), and with any values, valid or invalid.

We define the $Next$ relation by projecting the $System$ relation on the attributes of the global states. Note that all other attributes of the $System$ schema, be they input variables, output variables, or $DARMA$ variables, are implicitly existentially quantified by this construction.

$$Init == InitState$$
$$Next == \{System \bullet (\theta GlobalState, \theta GlobalState')\}$$

Given these, the set of traces and reachable states are defined straightforwardly.

$$Traces == \{f : \mathbb{N} \rightarrow GlobalState | f(0) \in (Init) \land (\forall i : \mathbb{N} \bullet (f(i), f(i+1)) \in Next)\}$$
$$ReachableStates == Next^*(\!|Init|\!)$$

5.3 Formalizing and Proving the Requirements

The architecture's informal requirements, given in Section 3.3, are phrased in terms of *events*, such as the system authenticating a user or generating a signature. One way to specify these requirements would be to associate these events with particular interface functions. While this is a natural way to think about the requirements, it leaves open the question of where these events are actually generated. In an operational specification, like our PROMELA specification, one can resolve this question by associating each event with the program points where the operation has committed. For example, an authentication might then refer to the point(s) where a user is authenticated successfully.

In our work here, we take an alternative approach that is less operational. We introduce abstract *event predicates* that characterize the *state changes* associated with events. This abstracts away from the control flow within interface functions. It also leads to a methodological shift

where, when we later refine our specification, we must then prove that the refinement produces the intended state changes. As a result, this approach removes one possibility of introducing errors during specification, which could arise from incorrectly identifying which program points lead to relevant state changes.

5.3.1 Requirement (1)

This requirement states that the signature architecture must authenticate a user before the user generates a hysteresis signature.

We begin by formalizing some auxiliary system event predicates. The first states that the signature log changes, for some user identifier, the second formalizes that the session table has been changed by a user logging in, and the third formalizes a change due to a user logging out.

$$
\begin{array}{|l}
siglogChanges : USER\_ID \rightarrow (GlobalState \leftrightarrow GlobalState) \\
\hline
\forall uid : USER\_ID \; ; \; s1, s2 : GlobalState \bullet \\
\quad (s1, s2) \in siglogChanges(uid) \\
\iff \\
\qquad ((uid \in \mathsf{dom}(s1.signature\_log) \land uid \in \mathsf{dom}(s2.signature\_log) \\
\qquad \land (s1.signature\_log(uid) \neq s2.signature\_log(uid))) \\
\quad \lor \\
\qquad ((uid \notin \mathsf{dom}(s1.signature\_log) \land uid \in \mathsf{dom}(s2.signature\_log))))
\end{array}
$$

$$
\begin{array}{|l}
userDoesLogin : USER\_ID \rightarrow (GlobalState \leftrightarrow GlobalState) \\
\hline
\forall uid : USER\_ID \; ; \; s1, s2 : GlobalState \bullet \\
\quad (s1, s2) \in userDoesLogin(uid) \\
\iff \\
\quad uid \notin \mathsf{dom}(s1.session\_table) \land uid \in \mathsf{dom}(s2.session\_table)
\end{array}
$$

$$
\begin{array}{|l}
userDoesLogout : USER\_ID \rightarrow (GlobalState \leftrightarrow GlobalState) \\
\hline
\forall uid : USER\_ID \; ; \; s1, s2 : GlobalState \bullet \\
\quad (s1, s2) \in (userDoesLogout(uid)) \\
\iff \\
\quad (uid \in \mathsf{dom}(s1.session\_table) \land uid \notin \mathsf{dom}(s2.session\_table))
\end{array}
$$

Our formalization of the first requirement is closely related to the LTL formula that we specified in our previous case-study. The relationship is this: our LTL formula formalized a *never claim*, which is the negation of the security property that we want to verify. Here we formulate the property positively. Also, as noted in Section 3.4, rather than formulating the property using temporal modalities like "before" and "until", we speak explicitly about time points and relations between them. So, in this example, the specification says that at every point where a user changes the signature log, there exists a previous time point where the user logged in. In other words, there must be a login for the user before the associated signature log entry is changed.

The property HSD_1a is then formalized as follows:

$$
\begin{array}{l}
\vdash\ \forall t : Traces\ ;\ n : \mathbb{N}\ ;\ uid : USER\_ID \bullet \\
\qquad\qquad (t\,n, t(n+1)) \in siglogChanges(uid) \\
\qquad \Rightarrow \\
\qquad\qquad (\exists k : 0\,..\,(n-1) \bullet \\
\qquad\qquad\quad (t\,k, t(k+1)) \in (userDoesLogin(uid)))
\end{array}
$$

This is the requirement as specified by Hitachi, which we also verified in the previous case study. Note however, that it is actually weaker than presumably was intended. In particular, the requirement admits traces where *userDoesLogin* occurs, followed by *userDoesLogout*, and later *siglogChanges*, e.g., possibly by an attempt to reuse an outdated session key generated by the previous login.

To eliminate such traces, we strengthen the formalization of this requirement further, additionally stipulating that a logout has not occurred between the login and the signature generation.

The property HSD_1b is presented as follows:

$$
\begin{array}{l}
\vdash\ \forall t : Traces\ ;\ n : \mathbb{N}\ ;\ uid : USER\_ID \bullet \\
\qquad\qquad (t\,n, t(n+1)) \in siglogChanges(uid) \\
\qquad \Rightarrow \\
\qquad\qquad (\exists k : 0\,..\,(n-1) \bullet (t\,k, t(k+1)) \in userDoesLogin(uid) \\
\qquad\qquad\quad \wedge (\forall j : (k+1)\,..\,(n-1) \bullet \\
\qquad\qquad\qquad (t\,j, t(j+1)) \notin userDoesLogout(uid)))
\end{array}
$$

5.3.2 Requirement (2)

The second requirement states that the architecture generates a hysteresis signature using the private key of an authenticated user. To formalize this, we essentially refine the previously defined event predicate *siglogChanges*.

$$
\begin{array}{|l}
siglogChangedTo : (USER\_ID \times SIGNATURE) \rightarrow \\
\qquad\qquad (GlobalState \leftrightarrow GlobalState) \\
\hline
\forall uid : USER\_ID\ ;\ sig : SIGNATURE\ ;\ s1, s2 : GlobalState \bullet \\
\quad (s1, s2) \in siglogChangedTo(uid, sig) \\
\iff \\
\qquad (((uid \in \mathsf{dom}(s1.signature\_log) \wedge uid \in \mathsf{dom}(s2.signature\_log) \\
\qquad \wedge (s1.signature\_log(uid) \neq s2.signature\_log(uid))) \\
\quad \vee \\
\qquad (uid \notin \mathsf{dom}(s1.signature\_log) \wedge uid \in \mathsf{dom}(s2.signature\_log))) \\
\quad \wedge s2.signature\_log(uid) = sig)
\end{array}
$$

$$
\begin{array}{|l}
signatureIsGenerated : (USER\_ID \times SIGNATURE) \rightarrow (GlobalState \leftrightarrow GlobalState) \\
\hline
\forall uid : USER\_ID\ ;\ sig : SIGNATURE\ ;\ s1, s2 : GlobalState \bullet \\
\quad (s1, s2) \in signatureIsGenerated(uid, sig) \\
\iff \\
\quad (\exists siglog : SIGNATURE\ ;\ hmg : \mathrm{seq}\, CHAR \bullet \\
\qquad uid \in \mathsf{dom}(s2.pri\_key\_list) \wedge \\
\qquad sig = hys\_sig\_gen(hmg, (s2.pri\_key\_list)(uid), siglog))
\end{array}
$$

The property HSD_2a can be directly formalized over traces in the following way.

$$
\begin{array}{l}
\vdash\ \forall t : Traces\ ;\ n : \mathbb{N}\ ;\ uid : USER\_ID\ ;\ sig : SIGNATURE \bullet \\
\qquad (t\,n, t(n+1)) \in siglogChangedTo(uid, sig) \\
\qquad \Rightarrow \\
\qquad\quad (\forall uid' : \mathsf{dom}(t\,n).signature\_log \setminus \{uid\} \bullet \\
\qquad\qquad ((t\,n).signature\_log)(uid') = \\
\qquad\qquad ((t\,(n+1)).signature\_log)(uid'))\wedge \\
\qquad\quad (t\,n, t(n+1)) \in signatureIsGenerated(uid, sig)
\end{array}
$$

This states that whenever a siglogChangedTo event occurs, *only* one signature has changed, and this change is in accordance with the *SignatureGeneration* method. Note that the formalization of this security requirement has in fact three aspects, which we tackled independently in the proof:

1. Whenever a *siglogChanges*-event for a user *uid* occurs, this user must occur in the domain of *PrivateKeyList*, since *pri_key_list*(*uid*) must be defined. This requires the global system invariant (holding for all traces) that the domain of the *Session_Table* is included in the domain of the *PrivateKeyList*.

2. A data integrity constraint holds for the generated signature. Namely, it must have been generated with the private key of *uid*. This amounts to a local postcondition of the *GenerateSignatureL*-schema.

3. We have strengthened Hitachi's second informal requirement to also require that the signature generated for *uid* leaves all other signatures unchanged ("... [only] by using the private key ..."). Without this constraint, it is possible to modify arbitrarily the signatures in the log during this system transition.

As an aside, note that in above security requirement, we only reason about consecutive events in a trace, i.e., a position and its successor. Thus an alternative formalization, given below, is to forgo traces and formalize the property in terms of reachable states, and their successors. In the end, which formulation one chooses, is mostly a matter of taste.

$$
\begin{array}{l}
\vdash\ \forall s1 : Next^*(\!|Init|\!) \bullet \\
\qquad \forall s2 : Next(\!|\{s1\}|\!)\ ;\ uid : USER\_ID \bullet \\
\qquad \exists sig : SIGNATURE \bullet \\
\qquad\quad (s1, s2) \in siglogChangedTo(uid, sig) \\
\qquad \Rightarrow \\
\qquad\quad (\forall uid' : \mathsf{dom}\ s1.signature\_log \setminus \{uid\} \bullet \\
\qquad\qquad ((s1.signature\_log)(uid')) = \\
\qquad\qquad ((s2.signature\_log)(uid')))\wedge \\
\qquad\quad (s1, s2) \in signatureIsGenerated(uid, sig)
\end{array}
$$

5.3.3 Requirement (3)

The third requirement states that the signature architecture generates only one signature per authentication. We interpret this requirement as follows: if two subsequent *siglogChangesTo* events occur (which must have resulted in generated signatures according to HSD_2), there must have been a logout event between them.

The property HSD_3 is presented as follows:

Basics:	elementary lemmas on types and data (defined in the *Basics*-section), which represents the "data dictionary" of our specification
SessionManager, **AccessController:**	auxiliary lemmas following from the definitions in these sections
HSD:	core theorems about our design specification, concerning the state of the system components and the component-local state transitions described as system operations
HSDArch:	architecture composition theorems associated with the architecture that is "wired together" in this section
Analysis:	behavioral theorems establishing general properties on system traces, general system invariants, and the proof of the system requirement properties of HSD_1–HSD_3

Figure 5: Proof scripts by Section

$$
\begin{array}{l}
\vdash\ \forall t : Traces\ ;\ n : \mathbb{N}\ ;\ uid : USER\_ID\ ;\ sig, sig' : SIGNATURE \bullet \\
\qquad (\forall k : 0\,..\,(n-1) \bullet \\
\qquad\quad (t\,k, t(k+1)) \in siglogChangedTo(uid, sig) \wedge \\
\qquad\quad (t\,n, t(n+1)) \in siglogChangedTo(uid, sig') \\
\qquad \Rightarrow \\
\qquad\quad (\exists j : (k+1)\,..\,(n-1) \bullet \\
\qquad\qquad (t\,j, t(j+1)) \in userDoesLogout(uid)))
\end{array}
$$

This formulation exploits the fact that HSD authenticates only one user at a time.

5.4 Proof Architecture and Selected Proof Details

The theorems proven, leading up to and including the proofs of HSD_1–HSD_3 (see Appendix B), were organized into separate theory files, following the structure of the specification: An SML file containing proof scripts is associated with each of the six Z-sections presented in Section 4). We summarize the kinds of theorems proven in Figure 5.

The theorems proven in the *Basics*-section are elementary lemmas that formalize simple properties of the defined data structures. For example, the lemma `X_in_SIGNATURE` states that any element of the type `VALID\_SIGNATURE + UNIT` is a signature. Other lemmas characterize basic integrity constraints on data structures. For example, `CRYPT_ERR_not_in_dom_dom_SESSION_TABLE` states that the session table does not contain results for session identifiers associated with operation errors. Such properties, although simple, are essential for later theorem proving: we extended Isabelle's automated proof procedures to employ these properties automatically during subsequent theorem proving.

The theories *SessionManager* and *AccessController* contains three kinds of theorems.

1. Simplification rules for the auxiliary functions. An example is the theorem `RegistSessionInformation_N`, shown below.

```
"[| uid ~: dom ssn_tbl ;
    ssn_tbl : SESSION_TABLE; ssn_IDs: %F SESSION_ID |] ==>
 RegistSessionInformation %^ (uid,ssn_tbl,ssn_IDs) = (new %^ ssn_IDs)";
```

2. Introduction and elimination rules for the predicates formalizing the security checks. Examples of such theorems are `isValidSession_2`, `isValidSession_3'`, `not_ReadPrivateKeyFailure1VSisValidSession` (i.e. if no read private key failure occurred, then there must be a valid session) and `AppendSignatureRecordFailure_VS_isValidSession`.

3. Key invariants of security checks. This includes theorems such as `CheckValidofSession_uid_in_dom_ssn_tbl1` (i.e. the user identifier returned by CheckValidofSession is in the domain of the session table) and `ReadPrivateKey_dom_session_table_inv` (i.e. ReadPrivateKey does not change the domain of the session table).

The complexity of these proofs ranges from trivial to moderate. Once proved, most of the facts were automatically used by Isabelle later. Note that the necessity of some of the invariance proofs was often not initially apparent and was only discovered later during the proof attempts of global proof invariants. For example, the condition in HSD_2a requiring that

$$((t\,n).signature\_log)(uid') = ((t\,(n+1)).signature\_log)(uid'))$$

for all uid' different from uid (for which the $siglogChanges$ was observed) initially appeared to be straightforward to prove. However, to prove this we needed to establish first the invariant `GenerateSignatureL_implies_not_siglogChanges` (in the HSD-Section), which in turn required the invariant `AppendSignatureRecord_imp_nosid_nochange` (in the AccessController-Section).

The theory of section *HSD* contains theorems about the preconditions, postconditions, and invariants of operator schemas, i.e., theorems of the following forms:

1. $OPSchema \Rightarrow COND \Rightarrow INV_{(\sigma,\sigma')}$, where the invariance INV is expressed in terms of (state variables from) the pre-state σ and the post-state σ', for example that a state variable doesn't change (i.e., $x = x'$ for some state variable x);

2. $OPSchema \Rightarrow COND \Rightarrow PRE_{\sigma}$, where the condition PRE is expressed in terms of the pre-state σ; and

3. $OPSchema \Rightarrow COND \Rightarrow POST_{\sigma'}$, where the condition $POST$ is expressed in terms of the post-state σ'.

Examples of the first kind of theorem are

- `AuthenticateUserL_inv_state_components`, stating that $AuthenticateUserL$ does not change the state components $signature\_log$,$access\_control\_list$, and pri_key_list;

- `GenerateSignatureL_siglog_mono`, stating that $GenerateSignatureL$ increases the domain of the $signature\_log$; and

- `GenerateSignatureL_and_siglogChanges_implies_inv_others`, stating that if $GenerateSignatureL$ is performed and if a siglogChanges occurred for uid, then the $signature\_log$ remains unchanged for all uid' different to uid.

An example of a theorem of the second type is `GenerateSignatureL_siglogChanges_charn`, stating that if $GenerateSignatureL$ is performed and if a siglogChange occurred for uid, uid must have been in $dom(session\_table)$ and it must have had a valid session identifier in the pre-state. Finally, an example for a theorem of the third type is `GenerateSignatureL_siglogChanges_charn2`. The complexity of these proofs ranges from trivial to very high, both in terms of the conceptual work required to understand why they hold as well as the effort required to carry out the proofs in Isabelle.

The theory of section *HSDArch* contains one main theorem, the *architecture decomposition theorem*, which states that the global system (consisting of arbitrary clients and the server communicating over DARMA) can make progress in exactly four ways:

1. a client makes a $AuthenticateUserW$ step in parallel with a $AuthenticateUserL$ step on the server side;
2. a client makes a $GenerateSignatureW$ step and the server a $NopOperationL$ step;
3. a client makes a $GenerateSignatureW$ step and the server a $GenerateSignatureL$; or
4. a client makes a $LogoutW$ step and the server a $LogoutL$-step.

By using the Z schema calculus, this theorem (`SysArch_introduction_theorem`) can be expressed in a surprisingly compact way, as follows.

$$\begin{aligned} \vdash \quad & (\exists DARMA \bullet AuthenticateUserW \land AuthenticateUserL) \lor \\ & (\exists DARMA \bullet GenerateSignatureW \land NopOperationL) \lor \\ & (\exists DARMA \bullet GenerateSignatureW \land GenerateSignatureL) \lor \\ & (\exists DARMA \bullet LogoutW \land LogoutL) \\ \Rightarrow \quad & System \end{aligned}$$

The theory of the *Analysis* starts with proofs of general properties about system traces that follow from the definition. Examples are:

- `traces_init_D1`, stating that for any system trace t, the components for $session\_table$, $signature\_log$ and $session\_ID$'s must initially be empty;
- `traces_init_D2`, stating that any system trace t must be a total function from natural numbers to global system states; and
- `trace_GlobalState`, stating that any global state in a trace satisfies the data invariants of $SessionManager$,$HysteresisSignature$ and $AccessController$.

The key theorem of this section is a reformulation of the architecture decomposition theorem `SysArch_introduction_theorem` as a rule for goal decompose by case-splitting: a property P over a system transition holds if it holds over one of the transitions caused by the four possible system transitions `State_Transition_Cases`. Based on this groundwork, two kinds of global invariants were established:

1. $t : traces \Rightarrow INV(t(n), t(n+1))$, i.e. for all positions n in a trace, invariant INV holds, and
2. $t : traces \Rightarrow (\forall\exists)^* n_1 \ldots n_m . INV(t, n_1, \ldots n_m)$, i.e. for all nestings of quantifiers over positions in a trace, invariant INV holds.

Note that invariants of the former class are a (pragmatically important) special case of the latter. Note, moreover, that formulas of the second class have the same expressive power as temporal logical formulas. While the desired overall proof goals HSD_1–HSD_3 are prominent members the second class, formulas of the first class comprise for example:

1. `acl_and_pkl_inv`, i.e. $access\_control\_list$ and $private\_key\_list$ never change;
2. `signature_log_mono`, i.e. the $signature\_log$ monotonically increases; and

Measurement	PROMELA/SPIN	HOL-Z/Isabelle
Model Size	647 lines	550 lines
Property Size	184 lines	50 lines
Proof Size	(None)	3662 lines
System Modeling Time	16 days	12 days
Property Specification Time	6 days	2 days
Verification Time	14,5 hours	19 days
Total Time	23 days	33 days

Figure 6: Quantitative Comparison of the Two Verifications

3. `dom_tabs_contained_in_dom_pri_key_list`, i.e. the domain of the *session_table* and the domain of the *signature_log* are always bounded by the domain of the *pri_key_list*.

These behavioral level proofs are typically established by routine induction over the position in a trace; after application of the architecture decomposition theorem `State_Transition_Cases` in the induction step, theorems of the data level (establishing that the postcondition of an operation schema implies an invariance over parts of traces) are used to reason about consecutive steps. Thus, the behavioral level can be seen (both syntactically and proof-technically) as an abstract interface for standard reasoning about pre- and postconditions.

6 Comparison and Conclusions

6.1 Comparison with Previous Case Study

Here we make both quantitative and qualitative comparisons with respect to our previous case study where we used the SPIN model checker to verify a PROMELA model of the HSD system. Note that such comparisons must be made and interpreted with care as the conclusions can differ considerably depending on the expertise of those involved in constructing the models and using the verification tools (see [**?**] for a discussion of these points). Still, we believe that our comparison sheds light on the relative strengths and weaknesses of the different approaches.

Figure 6 provides statistics[2] on the two approaches in terms of the size of the specifications and proofs and the time required to model the HSD system, its properties, and to carry out the verification (respectively model checking). We explain the different measurements below.

To begin with, despite their differences, both models are of roughly similar size. This stems from the fact that the HOL-Z model is more detailed than the PROMELA model in some regards and more abstract in others. For example, HOL-Z state schemas are more detailed since they not only define data types, but also invariants. On the other hand, HOL-Z operator schemas are typically smaller as they abstractly specify the relationship between states, rather than the operations used to change state. Also, communication is handled more abstractly in the HOL-Z model.

In contrast, the HOL-Z property (requirement) specifications are considerably more concise, due to their greater generality. In the PROMELA model, all of the relevant data domains (messages, keys, users, etc.) must be bounded in order to support finite-state model checking. Hence all statements quantifying over these sets must be translated into finite, but large, conjunctions

[2]Verification times are measured on a 3 gigaherz Pentium IV computer with 1 gigabyte of RAM. However, the compute time is basically only relevant for the SPIN verification. Isabelle requires 12 minutes in total to check all the proof scripts. Hence vast majority of the HOL verification time was due to human thinking and interaction.

and disjunctions. Moreover, rather than using event predicates as in HOL-Z, one must formulate changes in terms of explicit statements about program points as well as manipulated data. This too results in a more voluminous specification. So here we see one of the advantages of working with a general, behavioral model.

More time was spent in the theorem-proving approach than in the model-checking approach. The main difference is due to the fact that model checking is automatic (requiring here 6 hours of computation time) as opposed to interactive (the 19 days reflects the time spent interacting with the theorem prover). Folk wisdom is that, because of automation, model checking is much less time consuming than theorem proving. While this is indeed the case for the verification time itself, the *overall* time reduction, about 30%, is not so significant. Moreover, this difference is even less substantial when one takes into account that stronger theorems were proven in the HOL-Z approach (roughly 5 days were required for proving the stronger variants).

However, the numbers point only indirectly to what is probably the most interesting difference: *how* the time was spent. With SPIN, once the PROMELA model is constructed and the properties are specified, the (human) verification effort is focused on simplifying the problem so that the model checker terminates. This involves tuning constants as well as introducing abstractions and other simplifications. In some cases, the complexity of the model may even increase due to the addition of auxiliary variables, assertions, and new (monitor) processes. All of these additions were necessary during our verification. The time spent with these activities was significant and is reflected both in the increased time taken for system modeling and for property specification.

Note that these efforts are quite different from those required for verification in HOL-Z. There we built only one model. We neither had to work out any abstractions or restrictions in advance nor make subsequent changes during verification itself. Hence the specification time was shorter. In return, substantially more time was required for verification. Although some of this time was spent pushing low-level proof details through the Isabelle system, much of it concerned discovering, formalizing, and proving auxiliary system invariants, which were required to prove the properties of interest. As explained in Section 5.4, many of these invariants are interesting in their own right. Although discovering and proving invariants is a more time-consuming activity than (PROMELA) model simplification, it is certainly also a more insightful one. The insights not only led to a better understanding of why the properties hold, they also led to our discovering problems in the informal property specifications (they were too weak), which were not uncovered during the model-checking case study.

In both case studies, expert advice was necessary, albeit to a different degree, and during different phases of the formal method's application. In both approaches, it was possible to build the first model by a non-formal-methods expert, who received some training for the task. In the PROMELA/SPIN case study, subsequent modeling required moderate (although still fairly minimal) guidance, e.g., suggestions of how to carry out the different kinds of model simplifications, needed to reduce the search space. In the HOL-Z model, a general review and restructuring of the original specification leading to a more compact model presentation by an expert turned out to be advantageous. Finding suitably abstract formulations in Z appears to require a bit more expertise that finding "natural" formulations in PROMELA, which was perceived as a kind of programming language. With respect to property specification and verification, there were also differences. In the PROMELA/SPIN case study, we ended up representing HSD_3 by augmenting the model with auxiliary variables and a monitor process that tested the value of these variables. Subsequent verification was then automatic. In contrast, in HOL-Z, the formalization of the properties could be carried out be a trained "verification engineer", but theorem proving itself required considerable expertise. Finally, note that our experience with property specification discussed above, where the necessity of proving invariants helped us uncover specification weaknesses, suggests that the model-checking approach is more prone to undetected specification

errors. One possible way around this is would be to install a specification review phase in the model-checking process.

HOL-Z is a very rich language that allows for natural formalization of models at any desired level of abstraction. Our HOL-Z model is more general than the PROMELA model as it must neither commit to fixed, finite state spaces nor to particular data structures and algorithms for data manipulation. The model is more general and it leaves more flexibility in how the architecture can be refined. For example, in the Hitachi architecture, a user may only be logged into the system once, i.e., associated with one session. However, one could change the HSD-architecture into an architecture that supports multiple sessions per user (where a a user may login in multiple times, thereby acquiring multiple session identifiers) with fairly little effort: just three lines would change in the model, and the proofs for HSD_1a/b and, to a lesser extent, for HSD_2 and HSD_3 be hardly affected. Moreover, as noted above, the theorems verified are also more general; their strengthening came about in part from the fact that during theorem proving one is forced to think more about what is being proved. Finally, the invariants proved give us additional confidence in the model itself.

In contrast to the wide-spread belief that Z is limited primarily to data-oriented modeling, we have shown how Z can be used for behavioral modeling, too. Moreover, as a consequence of the fact that Z is semantically equivalently to higher-order Logic (HOL), Z can cope with behavioral modeling in an elegant way. A detailed look on the amount of proof work (both in time and lines of code) gives further evidence for this: many of these behavioral level proofs are typically standard inductions over the position in a trace. Thus, the behavioral level can be seen (both syntactically and proof-technically) as a simple abstract interface for the standard reasoning over pre- and postconditions, where most of the proof work is to be done.

A Ambiguities Uncovered During Specification and Proof

The following problems were uncovered during the formalization of the (informal) specification.

1. According to the original (informal) specification, the function $CheckValidofSession$ rewrites the session table even if the signature generation fails. We think it is an error of original specification. The table should not change when no signature is generated.

2. The formalization in HOL-Z forced us to think more carefully about our requirements and why they hold. This lead not only lead to the formalization of a number of system invariants, we also strengthened two of Hitachi's requirements, in particular requirements (1) and (2) as indicated above. In the first case, we strengthened the specification to also exclude the case of a user logging out before a change to the signature log. In the second case, we strengthen the specification to express that a signature log change *only* effects the entry associated with the specified user identifier, i.e., all other entries remain unchanged.

During the proving phase of HSD we uncovered the following ambiguity.

1. The informal specification requires exceptional elements in various data types (such as $NULL$ in $SIGNATURE$ or NO_USER in $USER\_ID$), and in our model we formalize these requirements directly. However, this raises the question: in which parts of the model should they be explicitly forbidden? In our formalization, we have excluded exceptional values communicated from DARMA. An alternative could be to include them in invariants of the system managers on the Linux side, like the session manager (which would result in blocking operations if exceptional values were every communicated). Our choice avoids blocking operations, but makes an assumption about DARMA communication that must be checked separately.

B Proofs

```
(* ************************************************************* *)
(*       Project          : HSD security analysis
          Author           : B. Wolff
          Affiliation      : ETH Z\"urich
          This theory      : Elementary proofs
                             (Version based on Latex-ZETA format)
          $Date: 2002/12/11 14:35:46 $
          $Revision: 1.3 $
          Release          : 2.5                                    *)
(* ************************************************************* *)

(* ISABELLE/HOL-Z START *)
(* toggle to sml-mode, start sml shell with start_holz *)
(*

cd "holz";
use_holz "Basics";

 *)
use "Prelude.ML";

fun pprint_off() = (print_depth 0; goals_limit:=0);
fun pprint_on() = (print_depth 50; goals_limit:=10);

pprint_off();
```

```
toToplevel Basics.axdefs;
toToplevel Basics.schemes;


Add_axdefs_TC (map snd Basics.axdefs);


Add_Univrule_TC(map (get_thm Basics.thy)
                    ["LOGOUT_RESULT_def","APPEND_SIGNATURE_RESULT_def",
                     "ACCESS_TYPE_def","SIG_LOG_WRITE_ACCESS_def",
                     "PRI_KEY_READ_ACCESS_def","COMMAND_def",
                     "USER_NAME_def","VALID_SIGNATURE_def",
                     "VALID_PRI_KEY_def","VALID_SESSION_ID_def",
                     "VALID_USER_ID_def","UNIT_def","CHAR_def","String_def"]);



(* making datatype induced theorems accessible to proof engine *)
val COMMAND_induct =
    get_thm Basics.thy "Basics.COMMAND_induct";
val PRI_KEY_READ_ACCESS_induct =
    get_thm Basics.thy "Basics.PRI_KEY_READ_ACCESS_induct";
val SIG_LOG_WRITE_ACCESS_induct =
    get_thm Basics.thy "Basics.SIG_LOG_WRITE_ACCESS_induct";
val ACCESS_TYPE_induct =
    get_thm Basics.thy "Basics.ACCESS_TYPE_induct";
val APPEND_SIGNATURE_RESULT_induct =
    get_thm Basics.thy "Basics.APPEND_SIGNATURE_RESULT_induct";
val LOGOUT_RESULT_induct =
    get_thm Basics.thy "Basics.LOGOUT_RESULT_induct";
val SESSION_ERROR_induct =
    get_thm Basics.thy "Basics.SESSION_ERROR_induct";


val authenticate_user_not_generate_signature=
    get_thm Basics.thy "Basics.authenticate_user_not_generate_signature";
val authenticate_user_not_logout=
    get_thm Basics.thy "Basics.authenticate_user_not_logout";
val authenticate_user_not_empty =
    get_thm Basics.thy "Basics.authenticate_user_not_empty";
val generate_signature_not_logout=
    get_thm Basics.thy "Basics.generate_signature_not_logout";
val generate_signature_not_empty=
    get_thm Basics.thy "Basics.generate_signature_not_empty";
val logout_not_empty=
    get_thm Basics.thy "Basics.logout_not_empty";


val accept_read_prikey_not_refuse_read_prikey=
    get_thm Basics.thy "Basics.accept_read_prikey_not_refuse_read_prikey";


val accept_write_siglog_not_refuse_write_siglog =
    get_thm Basics.thy "Basics.accept_write_siglog_not_refuse_write_siglog";


val read_prikey_not_read_siglog =
    get_thm Basics.thy "Basics.read_prikey_not_read_siglog";
val read_prikey_not_write_siglog =
    get_thm Basics.thy "Basics.read_prikey_not_write_siglog";
val write_siglog_not_read_siglog =
```

```
    get_thm Basics.thy "Basics.write_siglog_not_read_siglog";

val access_denied_not_sig_log_updated=
    get_thm Basics.thy "Basics.access_denied_not_sig_log_updated";

val invalid_session_id_error_not_session_terminated=
    get_thm Basics.thy "Basics.invalid_session_id_err_not_session_terminated";

val distincts = [authenticate_user_not_generate_signature,
             authenticate_user_not_logout,
             authenticate_user_not_empty,generate_signature_not_logout,
             generate_signature_not_empty,logout_not_empty,
             accept_read_prikey_not_refuse_read_prikey,
             accept_write_siglog_not_refuse_write_siglog,
             read_prikey_not_read_siglog,read_prikey_not_write_siglog,
             write_siglog_not_read_siglog,
             access_denied_not_sig_log_updated,
             invalid_session_id_error_not_session_terminated];

Addsimps    (distincts @ (map (fn X => X RS not_sym) distincts));
(* setup the prover with all distincts and symmetric variants *)


goal Basics.thy "!!X. X : %F SESSION_ID ==> (new %^ X) ~: X";
by(cut_facts_tac [new_axdef] 1);auto();
qed"new_fresh";
Addsimps[new_fresh];

goal Basics.thy "NULL = Inr unit";
by(cut_facts_tac [NO_USER_axdef] 1);auto();
qed"NULL_def";

goal Basics.thy "NULL_KEY = Inr unit";
by(cut_facts_tac [NO_USER_axdef] 1);auto();
qed"NULL_KEY_def";

goal Basics.thy "NO_USER = Inr unit";
by(cut_facts_tac [NO_USER_axdef] 1);auto();
qed"NO_USER_def";


goal Basics.thy "NULL : SIGNATURE";
by(cut_facts_tac [NO_USER_axdef] 1);auto();
qed"NULL_is_SIGNATURE";

goal Basics.thy "NULL_KEY : PRI_KEY";
by(cut_facts_tac [NO_USER_axdef] 1);auto();
qed"NULL_KEY_is_PRI_KEY";

goal Basics.thy "NO_USER : USER_ID";
by(cut_facts_tac [NO_USER_axdef] 1);auto();
qed"NO_USER_is_USER_ID";

Addsimps[NULL_is_SIGNATURE,NULL_KEY_is_PRI_KEY,NO_USER_is_USER_ID];
Delsimps[NULL_def,NULL_KEY_def,NO_USER_def];
(* These definitions should not be automatically unfolded,
   as is HOL-Z default strategy. *)


goal Basics.thy "CRYPT_ERR = Inr crypt_err";auto();
qed"CRYPT_ERR_def";
```

```
goal Basics.thy "NO_USER_ERR = Inr no_user_err";auto();
qed"NO_USER_ERR_def";

goal Basics.thy "INVALID_PW_ERR = Inr invalid_pw_err";auto();
qed"INVALID_PW_ERR_def";

goal Basics.thy "SAME_USER_ERR = Inr same_user_err";auto();
qed"SAME_USER_ERR_def";

Delsimps[CRYPT_ERR_def,NO_USER_ERR_def,SAME_USER_ERR_def];
(* correcting default configuration *)

goal Basics.thy
"AUTH_ERRORS =                                                \
\ {CRYPT_ERR, NO_USER_ERR, INVALID_PW_ERR, SAME_USER_ERR}";
auto();
qed"AUTH_ERRORS_def";



goal Basics.thy "CRYPT_ERR : AUTH_ERRORS";auto();
qed"CRYPT_ERR_in_AUTH_ERRORS";

goal Basics.thy "SAME_USER_ERR : AUTH_ERRORS";auto();
qed"SAME_USER_ERR_in_AUTH_ERRORS";

goal Basics.thy "NO_USER_ERR : AUTH_ERRORS";auto();
qed"NO_USER_ERR_in_AUTH_ERRORS";

goal Basics.thy "INVALID_PW_ERR : AUTH_ERRORS";auto();
qed"INVALID_PW_ERR_in_AUTH_ERRORS";

Delsimps[NO_USER_ERR_def,INVALID_PW_ERR_def,
         SAME_USER_ERR_def,AUTH_ERRORS_def];
(* correcting default configuration *)
Addsimps[CRYPT_ERR_in_AUTH_ERRORS,SAME_USER_ERR_in_AUTH_ERRORS,
         NO_USER_ERR_in_AUTH_ERRORS,INVALID_PW_ERR_in_AUTH_ERRORS];


goalw Basics.thy [SIGNATURE_def,sum_def,image_def] "X : SIGNATURE";
auto();by(res_inst_tac [("s","X")] sumE 1);auto();
qed"X_in_SIGNATURE";

goalw Basics.thy [USER_ID_def,sum_def,image_def] "X : USER_ID";
auto();by(res_inst_tac [("s","X")] sumE 1);auto();
qed"X_in_USER_ID";

goalw Basics.thy [PRI_KEY_def,sum_def,image_def] "X : PRI_KEY";
auto();by(res_inst_tac [("s","X")] sumE 1);auto();
qed"X_in_PRI_KEY";

goalw Basics.thy [SESSION_ID_def,sum_def,image_def] "X : SESSION_ID";
auto();by(res_inst_tac [("s","X")] sumE 1);auto();
qed"X_in_SESSION_ID";

Addsimps[X_in_SIGNATURE,X_in_SIGNATURE,X_in_USER_ID,X_in_PRI_KEY,
         X_in_SESSION_ID];
(* Any X of sum type is in the corresponding sets.
   Use this fact automatically. *)


goalw Basics.thy [SESSION_TABLE_def]
```

```
"!!ssn_tbl. ssn_tbl : SESSION_TABLE ==> NO_USER ~: dom ssn_tbl";
br contra_subsetD 1;
br Rel_Dom_subset 1;
be Pfun_Rel 1;
auto();
qed"NO_USER_not_in_dom_SESSION_TABLE";

goalw Basics.thy []
"!!ssn_tbl. [| ssn_tbl : SESSION_TABLE; x : dom ssn_tbl |] ==> x ~= NO_USER";
be contrapos2 1;back();
by(asm_full_simp_tac (simpset() addsimps [NO_USER_not_in_dom_SESSION_TABLE])1);
qed"NO_USER_not_in_dom_SESSION_TABLE_rev";
Addsimps[NO_USER_not_in_dom_SESSION_TABLE_rev];


goalw Basics.thy [SESSION_TABLE_def]
"!!ssn_tbl. [| ssn_tbl : SESSION_TABLE; x : dom ssn_tbl|] ==> \
\ CRYPT_ERR ~: dom(ssn_tbl %^ x)";
br contra_subsetD 1;
br Rel_Dom_subset 1;
br Pfun_Rel 1;
br pfun_apply 1; ba 1; ba 1;
by(auto_tac (claset(),simpset() addsimps [AUTH_ERRORS_def]));
qed"CRYPT_ERR_not_in_dom_dom_SESSION_TABLE";

goalw Basics.thy [SESSION_TABLE_def]
"!!ssn_tbl. [| ssn_tbl : SESSION_TABLE; x : dom ssn_tbl|] ==> \
\ NO_USER_ERR ~: dom(ssn_tbl %^ x)";
br contra_subsetD 1;
br Rel_Dom_subset 1;
br Pfun_Rel 1;
br pfun_apply 1; ba 1; ba 1;
by(auto_tac (claset(),simpset() addsimps [AUTH_ERRORS_def]));
qed"NO_USER_ERR_not_in_dom_dom_SESSION_TABLE";

goalw Basics.thy [SESSION_TABLE_def]
"!!ssn_tbl. [| ssn_tbl : SESSION_TABLE; x : dom ssn_tbl|] ==> \
\ INVALID_PW_ERR ~: dom(ssn_tbl %^ x)";
br contra_subsetD 1;
br Rel_Dom_subset 1;
br Pfun_Rel 1;
br pfun_apply 1; ba 1; ba 1;
by(auto_tac (claset(),simpset() addsimps [AUTH_ERRORS_def]));
qed"INVALID_PW_ERR_not_in_dom_dom_SESSION_TABLE";

goalw Basics.thy [SESSION_TABLE_def]
"!!ssn_tbl. [| ssn_tbl : SESSION_TABLE; x : dom ssn_tbl|] ==> \
\ SAME_USER_ERR ~: dom(ssn_tbl %^ x)";
br contra_subsetD 1;
br Rel_Dom_subset 1;
br Pfun_Rel 1;
br pfun_apply 1; ba 1; ba 1;
by(auto_tac (claset(),simpset() addsimps [AUTH_ERRORS_def]));
qed"SAME_USER_ERR_not_in_dom_dom_SESSION_TABLE";




goalw Basics.thy [ACCESS_CONTROL_LIST_def]
"!!acl. acl : ACCESS_CONTROL_LIST ==> NO_USER ~: dom acl";
br contra_subsetD 1;
br Rel_Dom_subset 1;
```

```
be Pfun_Rel 1;
auto();
qed"NO_USER_not_in_dom_ACCESS_CONTROL_LIST";
Addsimps[NO_USER_not_in_dom_SESSION_TABLE,
         CRYPT_ERR_not_in_dom_dom_SESSION_TABLE,
         NO_USER_ERR_not_in_dom_dom_SESSION_TABLE,
         INVALID_PW_ERR_not_in_dom_dom_SESSION_TABLE,
         SAME_USER_ERR_not_in_dom_dom_SESSION_TABLE,
         NO_USER_not_in_dom_ACCESS_CONTROL_LIST];



goal Basics.thy "!!X. X : %F SESSION_ID ==> (new %^ X) ~= CRYPT_ERR";
by(cut_facts_tac [new_axdef,AUTH_ERRORS_def] 1);
bd DECL_D1 1;
bd tfun_apply 1; ba 1;
auto();
qed"new_neq_CRYPT_ERR";

goal Basics.thy "!!X. X : %F SESSION_ID ==> (new %^ X) ~= NO_USER_ERR";
by(cut_facts_tac [new_axdef,AUTH_ERRORS_def] 1);
bd DECL_D1 1;
bd tfun_apply 1; ba 1;
auto();
qed"new_neq_NO_USER_ERR";

goal Basics.thy "!!X. X : %F SESSION_ID ==> (new %^ X) ~= INVALID_PW_ERR";
by(cut_facts_tac [new_axdef,AUTH_ERRORS_def] 1);
bd DECL_D1 1;
bd tfun_apply 1; ba 1;
auto();
qed"new_neq_INVALID_PW_ERR";

goal Basics.thy "!!X. X : %F SESSION_ID ==> (new %^ X) ~= SAME_USER_ERR";
by(cut_facts_tac [new_axdef,AUTH_ERRORS_def] 1);
bd DECL_D1 1;
bd tfun_apply 1; ba 1;
auto();
qed"new_neq_SAME_USER_ERR";
Addsimps [new_neq_CRYPT_ERR,new_neq_NO_USER_ERR,
          new_neq_INVALID_PW_ERR,new_neq_SAME_USER_ERR];


val prems = goalw Basics.thy [PRI_KEY_LIST_def]
"!! uid. [| X : dom pri_key_lst;                          \
\           pri_key_lst: PRI_KEY_LIST |] ==>              \
\   pri_key_lst %^ X ~= NULL_KEY";
by (forward_tac [pfun_apply] 1);ba 1;
auto();
qed"pri_key_lst_apply_not_NULL_KEY";
Addsimps [pri_key_lst_apply_not_NULL_KEY];


val prems = goalw Basics.thy [SESSION_TABLE_def]
"!!sid. [| ssn_tbl : SESSION_TABLE |] ==>                        \
\       {uid. uid : dom ssn_tbl & P uid}  <= USER_ID - {NO_USER}";
be pfunE 1;bd Rel_Dom_subset 1;
br subset_trans 1; ba 2;
auto();
qed"aux1";
```

```
 val prems = goalw Basics.thy [SESSION_TABLE_def]
 "!!sid.[| ssn_tbl : SESSION_TABLE;                                  \
 \         (sid, y) : X; (x, X) : ssn_tbl |]                         \
 \      ==>                                                          \
 \      { uid. uid : dom ssn_tbl & sid : dom (ssn_tbl %^             \
 \                                              uid)} ~= {}";
 by(asm_full_simp_tac (HOL_ss addsimps [all_not_in_conv RS sym]) 1);
 auto();
 qed "aux2";
(* The weird indentation is due to zeta: it does not accept the line:

% \    { uid. uid : dom ssn_tbl & sid : dom (ssn_tbl  %^  uid)} ~= {}";

   without the leading % !!!
 *)

val prems = goalw Basics.thy [SESSION_TABLE_def]
"!!sid.[| ssn_tbl : SESSION_TABLE;                                  \
\         sid : dom(ssn_tbl %^ x); x : dom ssn_tbl |]                 \
\      ==>                                                          \
\      { uid. uid : dom ssn_tbl & sid : dom (ssn_tbl %^             \
\                                              uid)} ~= {}";
by(asm_full_simp_tac (HOL_ss addsimps [all_not_in_conv RS sym]) 1);
by(res_inst_tac [("x","x")] exI 1);
auto();
qed "aux2'";


val prems = goalw Basics.thy [SESSION_TABLE_def]
"!!sid.[| ssn_tbl : SESSION_TABLE;                                  \
\         (sid, y) : X; (x, X) : ssn_tbl |]                         \
\       ==> {uid. uid : dom ssn_tbl & sid : dom (ssn_tbl %^         \
\                                                  uid)} \
\          <= dom ssn_tbl";
auto();
qed"aux3";

Add_Univdef_TC[X_in_SIGNATURE,X_in_USER_ID,
               X_in_PRI_KEY,X_in_SESSION_ID];


(* ********************************************************************** *)
(*      Project          : HSD security analysis
        Author           : B. Wolff
        Affiliation      : ETH Z\"urich
        This theory      : The Access Controller and
                           Hysteresis Signature Generator
                           (Version based on Latex-ZETA format)
        $Date: 2002/12/11 14:35:46 $
        $Revision: 1.3 $
        Release          : 2.5                                          *)
(* ********************************************************************** *)

(* cd "holz";
   use_holz "AccessController";

 *)

toToplevel AccessController.axdefs;
toToplevel AccessController.schemes;

Add_axdefs_TC (map snd AccessController.axdefs);
```

```
val prems = goal AccessController.thy
"[| uid ~: dom acc_cnt_lst ; ssn_tbl : SESSION_TABLE; ssn_IDs: %F SESSION_ID; \
\   hpw:seq CHAR; acc_cnt_lst : ACCESS_CONTROL_LIST |] ==>                    \
\ AuthenticateUser %^ (uid,hpw,acc_cnt_lst,ssn_tbl,ssn_IDs) = NO_USER_ERR";
by(cut_facts_tac ((AuthenticateUser_axdef RS DECL_D2)::prems) 1);
auto();
qed"AuthenticateUser_F1";
Addsimps[AuthenticateUser_F1];


val prems = goal AccessController.thy
"!! uid. [| uid : dom acl ; hpw ~= ?F(acl%^(uid));  sIDs: %F SESSION_ID;     \
\           hpw:seq CHAR; uid : dom ssn_tbl; ssn_tbl : SESSION_TABLE;        \
\           acl : ACCESS_CONTROL_LIST |] ==>  \
\ AuthenticateUser %^ (uid,hpw,acl,ssn_tbl,sIDs) = INVALID_PW_ERR";
by(cut_facts_tac [AuthenticateUser_axdef RS DECL_D2] 1);
by (REPEAT (etac ballE 1));
br trans 1;ba 1;
by(Asm_simp_tac 1);
by(stac if_P 1); ba 1;auto();
bind_thm("AuthenticateUser_F2",uresult());
(* trick proof that synthesizes projection condition that can
   not be parsed in this setting ... *)
Addsimps[AuthenticateUser_F2];


val prems = goal AccessController.thy
"!! uid. [| uid : dom acl ; hpw = ?F(acl%^(uid)); sIDs : %F SESSION_ID;     \
\   hpw:seq CHAR; ssn_tbl : SESSION_TABLE; acl : ACCESS_CONTROL_LIST |] ==> \
\ AuthenticateUser %^ (uid,hpw,acl,ssn_tbl,sIDs) =                          \
\ RegistSessionInformation %^ (uid,ssn_tbl,sIDs)";
by(cut_facts_tac [AuthenticateUser_axdef RS DECL_D2] 1);
by (REPEAT (etac ballE 1));
br trans 1;ba 1;
by(Asm_simp_tac 1);
by(stac if_not_P 1); back(); back();
auto();
bind_thm("AuthenticateUser_S1",uresult());
(* trick proof that synthesizes projection condition that can
   not be parsed in this setting ... *)
Addsimps[AuthenticateUser_S1];

(* we refine these rules further, i.e. we also unfold the
   RegistSessionInformation predicate up to the primitives ...
   And finally block the prover from automatically unfolding
   RegistSessionInformation such that no interference with
   the newly derived rules may occur. *)

val prems = goal AccessController.thy
"!! uid. [| uid : dom acl ; hpw = ?F(acl%^(uid)); ssn_IDs : %F SESSION_ID;  \
\           hpw:seq CHAR;  uid : dom ssn_tbl; ssn_tbl : SESSION_TABLE;      \
\           acl : ACCESS_CONTROL_LIST                          |] ==>       \
\ AuthenticateUser %^ (uid,hpw,acl,ssn_tbl,ssn_IDs) = SAME_USER_ERR";
auto();
by(stac if_not_P 1);
auto();
bind_thm("AuthenticateUser_F3",uresult());
Addsimps[AuthenticateUser_F3];

(* Enlists the conditions of success:
```

```
   - uid is defined in the access control list
   - the user is authenticated (the passwd phw matches the one
     stored in the acl)
   - uid is fresh (i.e. not already logged in)
 *)
val prems = goal AccessController.thy
"!! uid. [| uid : dom acl ; hpw = ?F(acl%^(uid));                          \
\           ssn_IDs : %F SESSION_ID; hpw:seq CHAR; uid ~: dom ssn_tbl;  \
\           ssn_tbl : SESSION_TABLE; acl : ACCESS_CONTROL_LIST |] ==>   \
\ AuthenticateUser %^ (uid,hpw,acl,ssn_tbl,ssn_IDs) = new %^ ssn_IDs";
auto();
by(stac if_not_P 1);
auto();
bind_thm("AuthenticateUser_Success",uresult());
Addsimps[AuthenticateUser_Success];

(* will no longer automatically unfold def of RegistSessionInformation *)
Delsimps[stripS(RegistSessionInformation_axdef RS DECL_D2)];



val prems = goal AccessController.thy
"!! uid. [| sid : SESSION_ID; ssn_tbl : SESSION_TABLE;       \
\           pri_key_lst: PRI_KEY_LIST |] ==>                 \
\   ((sid,read_prikey,ssn_tbl) ~: isValidSession_) -->       \
\   ((sid,ssn_tbl,pri_key_lst) : ReadPrivateKeyFailure_)";
auto();
qed"isValidSessionVSReadPrivateKeyFailure1";


val prems = goal AccessController.thy
"!! uid. [| (sid,ssn_tbl,pri_key_lst) ~: ReadPrivateKeyFailure_; \
\            sid : SESSION_ID; ssn_tbl : SESSION_TABLE;          \
\           pri_key_lst: PRI_KEY_LIST |] ==>                     \
\   ((sid,read_prikey,ssn_tbl) : isValidSession_)";
be swap 1;
br (isValidSessionVSReadPrivateKeyFailure1 RS mp) 1;
auto();
qed"not_ReadPrivateKeyFailure1VSisValidSession";



val [p1,p2,p3,p4] = goal AccessController.thy
"[| sid : SESSION_ID; ssn_tbl : SESSION_TABLE; pri_key_lst:      \
\   PRI_KEY_LIST;                                               \
\   fst (CheckValidofSession %^ (sid, read_prikey, ssn_tbl)) :   \
\           dom pri_key_lst |] ==>                              \
\                                                               \
\ ((sid,ssn_tbl,pri_key_lst) : ReadPrivateKeyFailure_) -->      \
\  ((sid,read_prikey,ssn_tbl) ~: isValidSession_) ";
by(cut_facts_tac [p1,p2,p3] 1);
auto();
by(rotate_tac ~3 1);
by(asm_full_simp_tac (simpset() addsimps [Let_def]) 1);
by(forward_tac [p4 RS pri_key_lst_apply_not_NULL_KEY] 1);
auto();
qed"isValidSessionVSReadPrivateKeyFailure2";



(* ReadSignatureRecord: Nothing to do.
```

```
   HOL-Z default setup will lead to automatic unfolds
   of ReadSignatureRecord and reduction to isValidSession,
   CheckValidofSession
   which will be treated by previously derived rules. *)
val prems = goal AccessController.thy
"!! uid. [| sid : SESSION_ID; ssn_tbl : SESSION_TABLE;                  \
\           sig_log: SIGNATURE_LOG; pri_key_lst: PRI_KEY_LIST;         \
\           (sid,read_siglog,ssn_tbl) ~: isValidSession_;              \
\           (sid,read_prikey,ssn_tbl) ~: isValidSession_ |] ==>        \
\   ((sid,ssn_tbl,pri_key_lst,sig_log) : ReadSignatureRecordFailure_)";
auto();
qed"isValidSessionVSReadSignatureRecordFailure1";



goalw Basics.thy [SESSION_TABLE_def]
" !!sid x.                                                             \
\       [| s_tab : SESSION_TABLE; sid : dom (gen_un (ran s_tab)) |]    \
\       ==> sid ~: AUTH_ERRORS";
by(asm_full_simp_tac (HOL_ss addsimps [partial_func_def,rel_def]) 1);
by(Blast_tac 1);
qed"aux4";


goalw Basics.thy [SESSION_TABLE_def]
"!!sid. [| s_tab : SESSION_TABLE; sid : dom (gen_un (ran s_tab)) |]    \
\        ==>  ? x. x : dom s_tab /\\  (sid : dom (s_tab %^ x))";
auto();
qed"aux0";



goalw AccessController.thy []
"!!sid.                                                                \
\        [| s_tab : SESSION_TABLE;                                     \
\           sid : dom (gen_un (ran s_tab)) |]                          \
\       ==> s_tab (+) {(SessionManager.choose %^ {y. y : dom s_tab /\\    \
\                                  sid : dom (s_tab %^ y)},            \
\                       {(sid, refuse_read_prikey,                     \
\                         snd(s_tab %^ (SessionManager.choose %^       \
\                             {y. y : dom s_tab /\\ sid : dom (s_tab %^ y)\
\                             }) %^ sid))                              \
\                       })}                                            \
\           : SESSION_TABLE";
by(simp_tac (HOL_ss addsimps [SESSION_TABLE_def]) 1);
br Partial_Func_overrid_Distr 1;
by (convert2hol_tac [SESSION_TABLE_def] 1);
by(Asm_simp_tac 1);
br conjI 1; br conjI 2;
by (convert2hol_tac [SESSION_TABLE_def] 3);
br choose_neq_NO_USER 1;
be aux4 3;
br PowI 1;
be aux1 1; ba 2;
by (forward_tac [aux0] 1); ba 1;
by(Step_tac 1);
by(eres_inst_tac [("Q","?X = {}")] contrapos2 1);
be aux2 1;
by(Blast_tac 1);
by(Blast_tac 1);
qed"aux5";
```

```
(* really ??? XXX *)
Delsimps[beta_apply_pfun,beta_apply_tfun, tfun_implies_pfun,tfun_apply,
         choose_in_X,choose_in_subset,choose_neq_NO_USER];

(* VERY TIME CONSUMING: TODO: optimize: Step 31 *)
goal AccessController.thy
"!!sid.[| s_tab : SESSION_TABLE; pkl : PRI_KEY_LIST;            \
\          ! x: dom s_tab. ! y: dom s_tab.                      \
\               (? s. s:dom(s_tab %^ x) & s:dom(s_tab %^ y))    \
\               =+=> (x = y);                                   \
\          sig_log : SIGNATURE_LOG; hms : seq CHAR |] ==>       \
\  (sid,s_tab,pkl,sig_log,hms)~:AppendSignatureRecordFailure_   \
\  =+=> ((sid, write_siglog,                                    \
\          snd(CheckValidofSession%^(sid,read_prikey,s_tab)))   \
\            : isValidSession_)";
by(stac (read_instantiate_sg (sign_of AccessController.thy)
         [("SignatureGeneration6","SignatureGeneration")]
         (stripS (AppendSignatureRecordFailure__axdef RS DECL_D2))) 1);
(* because of HOL-Z-Bug this is more complicated than necessary ... *)
by(stac (stripS(AppendSignatureRecord_axdef RS DECL_D2)) 6);
by(ALLGOALS(Asm_simp_tac));
by(ALLGOALS(asm_simp_tac (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def]
                                    addsplits [expand_if])));

br impI 1;
by(case_tac "sid ~: dom (gen_un (ran s_tab))" 1);
by(Asm_simp_tac 1);
by(cut_facts_tac [excl_mid] 1);
be disjE 1;
by(stac CheckValidofSession_F2 1);ba 2;
by(ALLGOALS(Asm_full_simp_tac));
be exE 1;

by(stac (stripS(CheckValidofSession_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(simp_tac (simpset() addsimps  [Let_def,asSet_def,image_def,maplet_def]
                                 addsplits [expand_if])1);
by(Asm_simp_tac 1);
br impI 1;
by (thin_tac "Ex ?X" 1); (* Cleanup *)

(* Unfold isValidSession, prove side-conditions. *)
by(stac (stripS (isValidSession__axdef RS DECL_D2)) 1);
by(res_inst_tac [("t","{?X}")] subst 3);
be aux5 4;
by(ALLGOALS(Asm_simp_tac));
by (convert2hol_tac [] 1);

(* Unfold CheckValidOfSession, prove side-conditions. *)
by(stac (stripS(CheckValidofSession_axdef RS DECL_D2)) 1);
by(res_inst_tac [("t","{?X}")] subst 3);
be aux5 4;
by(ALLGOALS(Asm_simp_tac));
by (convert2hol_tac [image_def] 1);

(* Unfold CheckValidOfSession, prove side-conditions. *)
by(stac if_P 1);
by(Blast_tac 1);
by(ALLGOALS(Asm_simp_tac));
```

```
(* now comes the main chunk: accept_write_siglog
   must be set in the updated table provided that
   accept_write_siglog was set in the original table. *)
by(simp_tac (simpset() addsimps  [Let_def,asSet_def,image_def, maplet_def]
                       addsplits [expand_if]) 1);
br conjI 1;
by(res_inst_tac [("x","SessionManager.choose %^ {y. y : dom s_tab & \
                      \ sid : dom (s_tab %^ y)}")] exI 1);
by(Asm_simp_tac 1);
by (convert2hol_tac [] 1);
by(eres_inst_tac [("Q","?X : isValidSession_")] contrapos2 1);
br isValidSession_3 1; ba 1;
by (convert2hol_tac [] 1);
by(ALLGOALS(Asm_full_simp_tac));
br allI 1;
br (disjCI RS (disj_commute RS iffD1)) 1;
br (disjCI RS (disj_commute RS iffD1)) 1;
by (Asm_full_simp_tac 1);
by(REPEAT (etac conjE 1));
by(eres_inst_tac [("Q","?X ~= ?Y")] contrapos2 1);
by (Asm_full_simp_tac 1);
by(stac choose_unique 1);
by(ALLGOALS(Asm_simp_tac));

br impI 1;
by(thin_tac "Ex ?X" 1);
br choose_neq_NO_USER 1;
by (Asm_full_simp_tac 1);

(* choose_neq_NO_USER has to preconditions that are proven in the
   following: A: proof of boundedness of choose-argument
              B: proof of non-emptiness of choose-argument *)
(* proof of boundedness *)
br pfunE 1;
by (asm_full_simp_tac (HOL_ss addsimps [SESSION_TABLE_def]) 1);
bd Rel_Dom_subset 1;
br subset_trans 1; ba 2;
auto();
br choose_in_subset 1;
br PowI 1;
br aux1 1;
br aux2' 2;
by (ALLGOALS Asm_simp_tac);
auto();

(* proof of ineptness *)
by(eres_inst_tac [("Q","?X = {}")] contrapos2 1);
by(asm_full_simp_tac (HOL_ss addsimps [all_not_in_conv RS sym]) 1);
by(res_inst_tac [("x","SessionManager.choose %^ {y. y : dom s_tab & \
                      \ sid : dom (s_tab %^ y)}")] exI 1);
by(Asm_simp_tac 1);
qed"AppendSignatureRecordFailure_VS_isValidSession";
(* a really cruel theorem !!! *)


goal AccessController.thy
"!!sid.[| s_tab : SESSION_TABLE; pkl : PRI_KEY_LIST |] ==>           \
\      dom (snd (CheckValidofSession %^ (sid,X,s_tab))) = dom s_tab";
by(stac (stripS(CheckValidofSession_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(case_tac "sid : dom (gen_un (ran s_tab))" 1);
```

```
by(ALLGOALS(Asm_simp_tac));
by(res_inst_tac [("ACCESS_TYPE","X")] ACCESS_TYPE_induct 1);
by(ALLGOALS(Asm_simp_tac));
by(ALLGOALS(asm_simp_tac (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def]
                                    addsplits [expand_if])));
by(auto_tac (claset(), simpset() addsimps
                      [asSet_def, choose_in_subset,aux1,aux2,aux3]));
qed"CheckValidofSession_dom_session_table_inv";
Addsimps[CheckValidofSession_dom_session_table_inv];


goal AccessController.thy
"!! sid. [|s_tab : SESSION_TABLE; pkl : PRI_KEY_LIST |] ==>          \
\dom (snd (ReadPrivateKey %^(sid,s_tab, pkl)))  = dom s_tab";
by(stac (stripS(ReadPrivateKey_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
qed"ReadPrivateKey_dom_session_table_inv";
Addsimps[ReadPrivateKey_dom_session_table_inv];



val prems = goal AccessController.thy
"!!sid. [| ssn_tbl : SESSION_TABLE; pri_key_lst:PRI_KEY_LIST;                \
\          hms : seq CHAR; sig_log: SIGNATURE_LOG |] ==>                     \
\       ((sid, ssn_tbl, pri_key_lst, sig_log, hms)                            \
\                   ~: AppendSignatureRecordFailure_)  =+=>                   \
\       (sid, write_siglog, ssn_tbl) : isValidSession_";
by(stac (stripS(AppendSignatureRecordFailure__axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
qed"notAppendSignatureRecordFailureVSisValidSession";


val [single_session, uid,x,sid,xsid,neq] = goal AccessController.thy
" [|  ! x:dom ssn_tbl.                                                  \
\             (! s. s : dom (ssn_tbl %^ x) --> dom(ssn_tbl %^  x) = {s}); \
\     uid : dom ssn_tbl;   x : dom ssn_tbl;                             \
\     sid : dom (ssn_tbl %^  uid);                                      \
\     xsid : dom (ssn_tbl %^  x);                                       \
\     sid ~= xsid |] ==>   uid ~= x";
by(cut_facts_tac [single_session, uid,sid,x,xsid,neq] 1);
be swap 1;
by(asm_full_simp_tac (simpset() addsimps [Ball_def]) 1);
by(eres_inst_tac [("x","uid")] all_dupE 1);
by(eres_inst_tac [("x","x")] allE 1);
by(hyp_subst_tac 1);
by(asm_full_simp_tac (simpset() addsimps []) 1);
by(eres_inst_tac [("x","sid")] all_dupE 1);
by(eres_inst_tac [("x","xsid")] allE 1);
be impE 1; ba 1;
be impE 1; ba 1;
by(Asm_full_simp_tac 1);
qed"single_session_implies_neqsids_implies_nequids";




goal AccessController.thy
"!! uid. \
\[| (uid : dom sig_log' & uid ~: dom sig_log) |                          \
```

```
\    (uid : dom signature_log & sig_log %^ uid ~= sig_log' %^ uid       \
\     );                                                               \
\     sig_log : SIGNATURE_LOG; acl : ACCESS_CONTROL_LIST;              \
\     pkl : PRI_KEY_LIST; ssn_tbl : SESSION_TABLE;                     \
\     (sid, read_prikey, ssn_tbl) : isValidSession_;                   \
\     (sid, write_siglog,  snd (ReadPrivateKey %^ (sid, ssn_tbl,pkl)   \
\      )):isValidSession_;                                             \
\     sig_log' = fst (snd (AppendSignatureRecord %^                    \
\                          (sid,                                       \
\                          snd (ReadPrivateKey %^ (sid, ssn_tbl, pkl)  \
\                              ),                                      \
\                          sig,                                        \
\                          sig_log)))                                  \
\  |] ==> fst (CheckValidofSession %^                                  \
\                (sid, write_siglog,                                   \
\                 snd (CheckValidofSession %^ (sid,read_prikey,ssn_tbl)  \
\                     ))) = uid";
auto();
by(defer_tac 1);
by(eres_inst_tac [("Pa","?X = ?Y")] swap 1);
by(stac (stripS (AppendSignatureRecord_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def]) 1);
br (CheckValidofSession_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def]) 1);
by(simp_tac (simpset() addsimps [maplet_def]) 1);
by(stac oplus_non_apply 1);
by(ALLGOALS(Asm_simp_tac));

br (neq_sym RS iffD1) 1; ba 1;
by(dres_inst_tac [("a","(uid, y)")] pair_rel_dom_fst 1);
by(eres_inst_tac [("Q","fst(uid,y) : ?X")] contrapos2 1);
by(ALLGOALS(Asm_full_simp_tac));
by(stac (stripS (AppendSignatureRecord_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def]) 1);
br (CheckValidofSession_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(ALLGOALS(Asm_simp_tac));
by(asm_full_simp_tac (simpset() addsimps [Let_def,maplet_def]) 1);
br (neq_sym RS iffD1) 1; ba 1;
qed"AppendSignatureRecord_lemma1";


(* This theorem covers the important case, that AppendSignatureRecord
   works without failure, but the user under consideration <uid>
   (having an active session) has a session identifier different to
   the one processed by AppendSignatureRecord. This means, it has to
   be shown that processing another user, say x with his session
   identifier xsid, does not change the session- and siglogtable for uid. *)

(* TIME CONSUMING: TODO: optimize *)
val [uidS,xS,sidUid,xsidx,neq,ssn_tbl,hmg,single_session,invert,sig_log,pkl,
     noReadPrivateKeyFailure,exec,noReadSignatureRecordFailure,
     noAppendSignatureRecordFailure] = goal AccessController.thy
"  [| uid : dom ssn_tbl;  x : dom ssn_tbl;                              \
\     sid : dom (ssn_tbl %^                                             \
\                uid);                                                  \
\     xsid : dom (ssn_tbl %^                                            \
\                 x);                                                  \
\     sid ~= xsid; ssn_tbl : SESSION_TABLE;hmg : seq CHAR;              \
\     ! x:dom ssn_tbl.(! s. s:dom(ssn_tbl %^ x                          \
```

```
\                                 )-->dom(ssn_tbl %^ x                      \
\                                                     )={s});               \
\      ! x:dom ssn_tbl. ! y:dom ssn_tbl.                                    \
\         (? s. s : dom (ssn_tbl %^ x) & s : dom (ssn_tbl %^ y              \
\               ))=+=> x = y;                                               \
\      sig_log : SIGNATURE_LOG;pkl : PRI_KEY_LIST;                          \
\      (xsid, ssn_tbl, pkl) ~: ReadPrivateKeyFailure_;                      \
\      (sig_log', ssn_tbl') = snd (AppendSignatureRecord %^                 \
\      (xsid, snd (ReadPrivateKey %^ (xsid, ssn_tbl, pkl)                   \
\                 ),                                                        \
\       SignatureGeneration %^                                              \
\          (xsid, ssn_tbl, pkl, sig_log,  hmg), sig_log));                  \
\      (xsid, ssn_tbl, pkl, sig_log) ~: ReadSignatureRecordFailure_;        \
\      (xsid, ssn_tbl, pkl, sig_log, hmg) ~: AppendSignatureRecordFailure_  \
\  |]  \
\  ==> ssn_tbl' %^ uid = ssn_tbl %^ uid & sig_log' %^ uid = sig_log %^ uid";

by(cut_facts_tac [ssn_tbl,sig_log,pkl] 1);
by(forward_tac [AppendSignatureRecordFailure_VS_isValidSession RS mp] 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps
                          [invert,sig_log,hmg,noAppendSignatureRecordFailure])));
br noAppendSignatureRecordFailure 2;  br hmg 1;
by (zftac (X_in_SESSION_ID RS (noReadPrivateKeyFailure RS
                               not_ReadPrivateKeyFailure1VSisValidSession)) 1);
by(zftac isValidSession_2' 1);
by(zftac isValidSession_3' 1);
by(REPEAT(etac conjE 1 ORELSE etac exE 1));
by(cut_facts_tac [exec] 1);
by(eres_inst_tac [("Q","(sig_log', ?X) = ?Y")] contrapos2 1);
by(zstac (AppendSignatureRecord_axdef RS DECL_D2) 1);
by(zstac (ReadPrivateKey_axdef RS DECL_D2) 1);
by(simp_tac (simpset() addsimps  [asSet_def,image_def,Let_def,maplet_def]
                       addsplits [expand_if])1);
by(Asm_simp_tac 1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);back();
by(ALLGOALS(asm_simp_tac (simpset() addsimps  [asSet_def,image_def,Let_def,maplet_def])));
by(stac if_not_P 1);
  by(Asm_simp_tac 1);
  by(res_inst_tac [("x","xa")] exI 1);
  by(Asm_simp_tac 1);
  by(zstac (CheckValidofSession_dom_session_table_inv RS sym) 1); ba 1;

by(Asm_simp_tac 1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
  by(ALLGOALS(asm_simp_tac (simpset() addsimps  [asSet_def,image_def,Let_def,maplet_def])));
  by(stac if_not_P 1);
  by(Blast_tac 1);
  by(Asm_simp_tac 1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps  [asSet_def,image_def,Let_def,maplet_def])));
  (* stupid type constraint reasoning: In
     ssn_tbl (+) {(choose ^ {y. y : dom ssn_tbl & xsid : dom (ssn_tbl ^ y)},
                         {(xsid, refuse_read_prikey,
                           (ssn_tbl ^ (choose ^ {y.
             y : dom ssn_tbl &
             xsid : dom (ssn_tbl ^ y)}) ^ xsid).slwa)})}
           : SESSION_TABLE
     nothing works automatic since non-determinism of choose must
     be checked against typ-constraint
  *)

  by(asm_full_simp_tac (HOL_ss addsimps [SESSION_TABLE_def,oplus_pfunI]) 1);
```

```
  br oplus_pfunI 1; ba 1;
  br pair_pfunI 1;
  br choose_in_subset 1; br PowI 1; br aux1 1;
  by(asm_full_simp_tac (HOL_ss addsimps [SESSION_TABLE_def]) 1);
  br aux2' 1;
  by(asm_full_simp_tac (HOL_ss addsimps [SESSION_TABLE_def]) 1);
  ba 1; ba 1; br aux1 1;
  by(asm_full_simp_tac (HOL_ss addsimps [SESSION_TABLE_def]) 1);
  br pair_pfunI 1;
  by(res_inst_tac [("f","ssn_tbl %^ x\
                                    \")] (Pfun_Rel RS Rel_Dom_Elem) 1);
  ba 2;
  be(pfun_apply) 1; ba 1;
  by(asm_simp_tac Z2HOL_ss 1);
  by(tc_tac 1);
(* <<<<<<<<<<<<<<<<<<<<<<<<<<< *)
by(stac if_P 1);
by(stac if_not_P 2);
by(ALLGOALS Asm_simp_tac);
(* Factoring out common subexpressions choose ^ {y. y : dom ssn_tbl & ... } *)
by(res_inst_tac [("s","x"),("t","SessionManager.choose %^ \
                                                    \ ?X")] subst 3);
back(); back();
by(res_inst_tac [("s","x"),("t","SessionManager.choose %^ \
                                                    \ ?X")] subst 4);
by(res_inst_tac [("s","x"),("t","SessionManager.choose %^ \
                                                    \ ?X")] subst 5);

by(eres_inst_tac [("Pa"," ?X & ?Y")] swap 6);
by(HINT "uid ~= x" (K ((rtac (single_session RS
                              single_session_implies_neqsids_implies_nequids) 7)
                       THEN (rtac neq 11)
                       THEN (ALLGOALS (simp_tac (HOL_ss addsimps
                            [uidS,sidUid,xS,xsidx,neq])))))) 6);
by(Asm_full_simp_tac 6); (* that's it: if uid is not x; then
                            all updates done matter only for x (but
                            not for uid). *)
(* the left-overs ... *)
by(res_inst_tac [("s","x"),("t","SessionManager.choose %^ \
                                                    \?X")] subst 1);
  br ([ssn_tbl,invert] MRS (choose_unique RS sym)) 1; ba 1; ba 1;
  by(Blast_tac 1);

br exI 1; br conjI 1; br disjI1 1; br refl 1;
  by(res_inst_tac [("s","x"),("t","SessionManager.choose %^ \
                                                      \?X")] subst 1);
  br ([ssn_tbl,invert] MRS (choose_unique RS sym)) 1; ba 1; ba 1;
  by(ALLGOALS Asm_simp_tac);
  by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
  by(asm_simp_tac (simpset() addsimps
                            [asSet_def,image_def,Let_def,maplet_def])1);
  by(stac if_not_P 1);
    by(Blast_tac 1);
    by(Asm_simp_tac 1);
      by(res_inst_tac [("s","x"),("t","SessionManager.choose %^ \
                                                         \?X")] subst 1);
      br ([ssn_tbl,invert] MRS (choose_unique RS sym)) 1; ba 1; ba 1;
      by(stac oplus_by_pair_apply1  1); by(Asm_simp_tac 2);
      by(thin_tac "~(?X & ?Y)" 1);
      br (((invert RS bspec) RS bspec) RS mp) 1;
      by(asm_full_simp_tac (HOL_ss addsimps [CheckValidofSession_dom_session_table_inv]) 1);
      ba 1;
```

```
      by(res_inst_tac [("x","xsid")] exI 1);
      by(Asm_full_simp_tac 1);
      bd(sid_ind_dom_CheckValidofSession_inv RS iffD1) 1;
      by(ALLGOALS Asm_simp_tac);
      br invert 1;

by(ALLGOALS(thin_tac "~(?X & ?Y)" ));
by(ALLGOALS(thin_tac "accept_write_siglog = ?X" ));
by(ALLGOALS(thin_tac "accept_read_prikey = ?X" ));
br ([ssn_tbl,invert] MRS (choose_unique RS sym)) 1; ba 1; ba 1;
by(ALLGOALS(rtac (choose_unique' RS sym)));
bd NO_USER_not_in_dom_SESSION_TABLE_rev 4; ba 4; ba 4;
bd NO_USER_not_in_dom_SESSION_TABLE_rev 2; ba 2; ba 2;
by(ALLGOALS(rtac (set_ext)));
by(ALLGOALS Asm_simp_tac);

by(ALLGOALS(rtac iffI));
by(ALLGOALS Asm_simp_tac);
by(REPEAT(etac conjE 1));
by(REPEAT(etac conjE 2));
be disjE 2;
by(ALLGOALS Asm_simp_tac);
by(ALLGOALS (eres_inst_tac [("P","xb : dom ssn_tbl")] rev_mp) ); (* to make subgoals equal *)
by(distinct_subgoals_tac);
br impI 1;
by(case_tac "xb = x" 1);
by(rotate_tac ~1 2);
by(ALLGOALS Asm_full_simp_tac);
be swap 1;
br (((invert RS bspec) RS bspec) RS mp) 1;
auto();
qed"AppendSignatureRecord_imp_nosid_nochange";


goal AccessController.thy
"!! uid. \
\[| (uid : dom sig_log' & uid ~: dom sig_log) |                          \
\    (uid : dom signature_log & sig_log %^ uid ~= sig_log' %^ uid);      \
\     sig_log : SIGNATURE_LOG; acl : ACCESS_CONTROL_LIST;                \
\     pkl : PRI_KEY_LIST; ssn_tbl : SESSION_TABLE;                       \
\     (sid, read_prikey, ssn_tbl) : isValidSession_;                     \
\     (sid, write_siglog,                                                \
\          snd (ReadPrivateKey %^ (sid, ssn_tbl,pkl))):isValidSession_; \
\     ! x:dom ssn_tbl. ! y:dom ssn_tbl.                                  \
\           (? s. s:dom (ssn_tbl %^ x) & s:dom (ssn_tbl %^ y))=+=> x=y; \
\     sig_log' = fst (snd (AppendSignatureRecord %^                      \
\                          (sid,                                         \
\                          snd (ReadPrivateKey %^ (sid, ssn_tbl, pkl)    \
\                              ),                                        \
\                          sig,                                          \
\                          sig_log)))                                    \
\  |] ==> fst(CheckValidofSession%^(sid,read_prikey,ssn_tbl)             \
\            ) = uid";
by(zftac isValidSession_2' 1);
by(zftac isValidSession_3' 1);
by(eres_inst_tac [("Q","sig_log' = ?Y")] contrapos2 1);
by(zstac (AppendSignatureRecord_axdef RS DECL_D2) 1);
by(zstac (ReadPrivateKey_axdef RS DECL_D2) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps  [Let_def,maplet_def]
                                    addsplits [expand_if])));
be conjE 1;
by(eres_inst_tac [("Q","(sid,write_siglog,?X) : ?Y")] contrapos2 1);
```

```
by(zstac (ReadPrivateKey_axdef RS DECL_D2) 1);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps [Let_def])));
be swap 1;
by(rotate_tac ~1 1);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps [Let_def])));
by(REPEAT (etac conjE 1));
by(hyp_subst_tac 1);

be disjE 1;
by(REPEAT (etac conjE 1));
by(REPEAT (etac conjE 2));

by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 2);
by(ALLGOALS(asm_full_simp_tac
            (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])));

br choose_unique 1;
br choose_unique 5;
by(ALLGOALS(Asm_simp_tac));
be disjE 2;be disjE 1;
by(Blast_tac 4);
by(Blast_tac 2);
by(hyp_subst_tac 1);by(hyp_subst_tac 2);

(* one: fst (CheckValidofSession %^ (sid, write_siglog,
  snd (CheckValidofSession %^ (sid, read_prikey, ssn_tbl))))  : dom ssn_tbl *)
be exE 1;back();
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_full_simp_tac
            (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(stac if_not_P 1);
by(Asm_simp_tac 1);
by(Asm_simp_tac 2);
by(res_inst_tac [("x","x")] exI 1);
by(Asm_simp_tac 1);
br choose_in_subset 1;
br PowI 1;
be aux1 1;
by(Asm_simp_tac 1);
by(asm_full_simp_tac (HOL_ss addsimps [all_not_in_conv RS sym]) 1);
by(Blast_tac 1);
by(Blast_tac 1);

(* two:  sid : dom (ssn_tbl %^ fst (CheckValidofSession %^ (sid, write_siglog,
                    snd (CheckValidofSession %^ (sid, read_prikey,
          ssn_tbl))))) *)
be exE 1;
by(REPEAT (etac conjE 1));
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_full_simp_tac
            (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_full_simp_tac
            (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(stac if_not_P 1);
by(Blast_tac 1);
by(res_inst_tac[("s","x")] (choose_unique' RS ssubst) 1);
br set_ext 1;
br iffI 1;
by(ALLGOALS(Asm_full_simp_tac));
by(res_inst_tac[("s","x")] (choose_unique' RS ssubst) 2);
```

```
by(asm_simp_tac (simpset() addsimps [Overrid_Domain]) 4);
by(ALLGOALS(Asm_simp_tac));
be conjE 1;
by(eres_inst_tac [("Q","sid : dom ((ssn_tbl (+) ?X) %^ xa) ")] contrapos2 1);
by(res_inst_tac[("s","x")] (choose_unique' RS ssubst) 1);
by(Blast_tac 4);
by(Blast_tac 1);
by(Asm_simp_tac 1);
by(stac oplus_by_pair_apply2 1); ba 1;
by(eres_inst_tac [("x","x")] ballE 1);
by(eres_inst_tac [("x","xa")] ballE 1);
by(Blast_tac 2);
by(Blast_tac 2);
by(Blast_tac 1);

(* three:   uid : dom signature_log;uid ~: dom ssn_tbl ==>
          sig_log %^ uid =
          (sig_log (+) {(fst (CheckValidofSession %^ (sid, write_siglog,
                snd (CheckValidofSession %^ (sid, read_prikey, ssn_tbl)))),
                        sig)}) %^ uid   *)
by(eres_inst_tac [("Pa","sig_log %^ uid = ?X")] swap 1);
be exE 1;
by(REPEAT (etac conjE 1));
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_full_simp_tac
            (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(stac oplus_by_pair_apply2 1); br refl 2;
br (neq_sym RS iffD1) 1;
br choose_neq_X 1;
by(Blast_tac 3);
br PowI 1;
be aux1 1;
by(simp_tac (HOL_ss addsimps [all_not_in_conv RS sym]) 1);
by(res_inst_tac [("x","x")] exI 1);
by(Asm_simp_tac 1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_simp_tac
            (simpset() addsplits [expand_if]
                       addsimps  [Let_def,asSet_def,image_def,maplet_def])1);
br impI 1;
by(stac oplus_by_pair_apply1 1);
br sym 1;
br choose_unique 1;
by(ALLGOALS(Asm_simp_tac));


(* four:   uid : dom signature_log;sid ~: dom (ssn_tbl %^ uid) ==>
          sig_log %^ uid =
          (sig_log (+) {(fst (CheckValidofSession %^ (sid, write_siglog,
                snd (CheckValidofSession %^ (sid, read_prikey, ssn_tbl)))),
                        sig)}) %^ uid  *)
by(eres_inst_tac [("Pa","sig_log %^ uid = ?X")] swap 1);
be exE 1;
by(REPEAT (etac conjE 1));
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_full_simp_tac
            (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);

by(stac oplus_by_pair_apply2 1); br refl 2;
br (neq_sym RS iffD1) 1;
br choose_neq_X 1;
br PowI 1;
```

```
be aux1 1;
by(simp_tac (HOL_ss addsimps [all_not_in_conv RS sym]) 1);
by(res_inst_tac [("x","x")] exI 1);
by(Asm_simp_tac 1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_simp_tac
           (simpset() addsplits [expand_if]
                      addsimps  [Let_def,asSet_def,image_def,maplet_def])1);
br impI 1;
by(stac oplus_by_pair_apply1 1);
br sym 1;
br choose_unique 1;
by(ALLGOALS(Asm_simp_tac));
br disjI2 1;
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_simp_tac
           (simpset() addsplits [expand_if]
                      addsimps  [Let_def,asSet_def,image_def,maplet_def])1);
br impI 1;
by(stac oplus_by_pair_apply2 1);
by(ALLGOALS(Asm_simp_tac));
br (neq_sym RS iffD1) 1;
br choose_neq_X 1;
br PowI 1;
be aux1 1;
by(Blast_tac 2);
by(simp_tac (HOL_ss addsimps [all_not_in_conv RS sym]) 1);
by(res_inst_tac [("x","x")] exI 1);
by(Asm_simp_tac 1);
qed"AppendSignatureRecord_lemma2";
(* a really cruel lemma. With all bells and whistles over CheckValidofSession.
   And what you never wanted to know about it ... ;-) *)


(* ********************************************************************** *)
(*      Project          : HSD security analysis
        Author           : B. Wolff
        Affiliation      : ETH Z\"urich
        This theory      : The Session Manager -
                           (Version based on Latex-ZETA format)
        $Date: 2002/12/11 14:35:46 $
        $Revision: 1.3 $
        Release          : 2.5                                              *)
(* ********************************************************************** *)

(* cd "holz";
   use_holz "SessionManager";
*)


toToplevel SessionManager.axdefs;
toToplevel SessionManager.schemes;

Add_axdefs_TC (map snd SessionManager.axdefs);




(* ********************************************************************** *)
(* Basics on RegistSessionInformation                                     *)
(* ********************************************************************** *)

val prems = goal SessionManager.thy
```

```
"[| uid : dom ssn_tbl ; ssn_tbl : SESSION_TABLE ; ssn_IDs: %F SESSION_ID|] ==>  \
\ RegistSessionInformation %^ (uid,ssn_tbl,ssn_IDs) = SAME_USER_ERR";
by(cut_facts_tac ((RegistSessionInformation_axdef RS DECL_D2)::prems) 1);
auto();
qed "RegistSessionInformation_F";
Addsimps[RegistSessionInformation_F];


val prems = goal SessionManager.thy
"[| uid ~: dom ssn_tbl ; ssn_tbl : SESSION_TABLE; ssn_IDs: %F SESSION_ID |] ==>  \
\ RegistSessionInformation %^ (uid,ssn_tbl,ssn_IDs) = (new %^ ssn_IDs)";
by(cut_facts_tac ((RegistSessionInformation_axdef RS DECL_D2)::prems) 1);
auto();
qed "RegistSessionInformation_N";
Addsimps[RegistSessionInformation_N];
(* HERE *)


val prems = goal SessionManager.thy
"[| uid ~: dom ssn_tbl ; ssn_tbl : SESSION_TABLE; ssn_IDs: %F SESSION_ID |] ==>  \
\ RegistSessionInformation %^ (uid,ssn_tbl,ssn_IDs) ~: AUTH_ERRORS";
by(cut_facts_tac (prems) 1);
by(stac RegistSessionInformation_N 1);
auto();
bd((new_axdef RS DECL_D1) RS tfun_apply) 1;
auto();
qed "RegistSessionInformation_NOERROR_INV";
Addsimps[RegistSessionInformation_NOERROR_INV];



(* ********************************************************************** *)
(* Basics on FreeSession                                                  *)
(* ********************************************************************** *)

goal SessionManager.thy
"!!X.  [| X : %P (USER_ID - {NO_USER}); X ~= {} |] ==> \
\         ((SessionManager.choose %^ X) : X)";
by(cut_facts_tac [choose_axdef RS DECL_D2] 1);
auto();
qed"choose_in_X";
Addsimps[choose_in_X];


goal SessionManager.thy
"!!X.  [| X : %P (USER_ID - {NO_USER}); X ~= {}; X <= Y |] ==> \
\         ((SessionManager.choose %^ X) : Y)";
bd choose_in_X 1;ba 1;auto();
qed"choose_in_subset";
Addsimps[choose_in_subset];



goal SessionManager.thy
"!!X. [| X : %P (USER_ID - {NO_USER}); X ~= {} |] ==> \
\((SessionManager.choose %^ X) ~= NO_USER)";
by(forward_tac [choose_in_X] 1); ba 1;
be swap 1;
by(rotate_tac ~1 1);
by(Asm_full_simp_tac 1);
auto();
qed"choose_neq_NO_USER";
```

```
Addsimps[choose_neq_NO_USER];

goal SessionManager.thy
"!!X. [| X : %P (USER_ID - {NO_USER}); X ~= {}; x ~:X |] ==> \
\((SessionManager.choose %^ X) ~= x)";
bd choose_in_X 1; ba 1;auto();
qed"choose_neq_X";



goal SessionManager.thy
"!!sid.                                                          \
\   [| s_tab : SESSION_TABLE;                                    \
\      ! x:dom s_tab.                                            \
\          ! y:dom s_tab.                                        \
\           (? s. s:dom (s_tab %^ x) & s:dom (s_tab %^ y))=+=>    \
\           x = y;                                               \
\      xa : dom s_tab;  sid : dom (s_tab %^ xa)|]                \
\   ==> (SessionManager.choose %^                                \
\         {y. y : dom s_tab &  sid : dom (s_tab %^ y)            \
\                                                  }) = xa ";
by(eres_inst_tac [("x","xa")] ballE 1);
by(Asm_full_simp_tac 2);
be ballE 1; be impE 1; be sym 2;
br exI 1; br conjI 1; ba 1;
by (HINT "{y. (y : dom s_tab) & sid : (dom (s_tab %^ y))         \
\         } :  %P(USER_ID - {NO_USER})"
        (K all_tac) 1);
bd choose_in_X 1;
by(asm_full_simp_tac (simpset() addsimps []) 2);
br PowI 2;
be aux1 2;
by(eres_inst_tac [("Pa"," ?X %^ ?Y: ?Z")] swap 2);
br choose_in_subset 2;
by(Blast_tac 4);
br PowI 2;
be aux1 2;
by(Step_tac 1);
by(eres_inst_tac [("Q","?X = {}")] contrapos2 1);
by(eres_inst_tac [("Q","?X = {}")] contrapos2 2);
be aux2' 1;
be aux2' 3;
auto();
qed"choose_unique";


goal SessionManager.thy
"!!X.  [| X = {x}; x ~= NO_USER |] ==> (SessionManager.choose %^ X) = x ";
by(cut_facts_tac [choose_axdef RS DECL_D2] 1);
by(eres_inst_tac [("x","{x}")] ballE 1);
auto();
qed"choose_unique'";



val prems = goal SessionManager.thy
"[| sid : dom (gen_un (ran ssn_tbl)); ssn_tbl : SESSION_TABLE |] ==>            \
\ FreeSessionInformation %^ (sid,ssn_tbl) = (session_terminated, ?X)";
by(cut_facts_tac ((FreeSessionInformation_axdef RS DECL_D2)::prems) 1);
by (REPEAT (etac ballE 1));
br trans 1;ba 1;auto();
qed "FreeSessionInformation_N";
```

```
Addsimps[FreeSessionInformation_N];


val prems = goal SessionManager.thy
"[| sid ~: dom (gen_un (ran ssn_tbl)); ssn_tbl : SESSION_TABLE |] ==>   \
\ FreeSessionInformation %^ (sid,ssn_tbl) = (invalid_session_id_err, ssn_tbl)";
by(cut_facts_tac ((FreeSessionInformation_axdef RS DECL_D2)::prems) 1);
by (REPEAT (etac ballE 1));
br trans 1;ba 1;auto();
qed "FreeSessionInformation_F";
Addsimps[FreeSessionInformation_F];



val prems = goal SessionManager.thy
"!!sid. [| sid : SESSION_ID; ssn_tbl : SESSION_TABLE |] ==>             \
\ sid ~: dom (gen_un (ran (snd(FreeSessionInformation %^ (sid,ssn_tbl)))))";
by(case_tac "sid : dom (gen_un (ran ssn_tbl))" 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps
                          [asSet_def,image_def,maplet_def])));
auto();
qed "FreeSessionInformation_deletes_sid1";
Addsimps[FreeSessionInformation_deletes_sid1];



val prems = goal SessionManager.thy
"!!sid. [| uid : dom ssn_tbl ; sid : dom (ssn_tbl %^ uid); ssn_tbl : SESSION_TABLE |] ==>
\
\   uid ~: dom(snd(FreeSessionInformation %^ (sid,ssn_tbl)))";
by(case_tac "sid : dom (gen_un (ran ssn_tbl))" 1);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                          [asSet_def,image_def,maplet_def,SESSION_TABLE_def])));
auto();
qed "FreeSessionInformation_deletes_sid2";
Addsimps[FreeSessionInformation_deletes_sid2];

val prems = goal SessionManager.thy
"!!sid. [| uid : dom ssn_tbl ; sid ~: dom (ssn_tbl %^ uid); ssn_tbl : SESSION_TABLE |] ==>
\
\   uid : dom(snd(FreeSessionInformation %^ (sid,ssn_tbl)))";
by(case_tac "sid : dom (gen_un (ran ssn_tbl))" 1);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                          [asSet_def,image_def,maplet_def,SESSION_TABLE_def])));
auto();
qed "FreeSessionInformation_deletes_sid3";
Addsimps[FreeSessionInformation_deletes_sid3];



(* remove automatic unfolding of CheckValidofSession_axdef. *)
Delsimps[stripS(CheckValidofSession_axdef RS DECL_D2)];

val prems = goal SessionManager.thy
"[| sid ~: dom (gen_un (ran ssn_tbl)); ssn_tbl : SESSION_TABLE |] ==>   \
\ CheckValidofSession %^ (sid,X,ssn_tbl) = (NO_USER, ssn_tbl)";
by(cut_facts_tac ((CheckValidofSession_axdef RS DECL_D2)::prems) 1);
by (REPEAT (etac ballE 1));
br trans 1;ba 1;auto();
qed "CheckValidofSession_F1";
Addsimps[CheckValidofSession_F1];
```

```
val prems = goal SessionManager.thy
"!!sid. \
\[| sid : dom (gen_un (ran ssn_tbl)); ! x. x ~: dom ssn_tbl | ?X sid x;\
\   ssn_tbl : SESSION_TABLE |] ==>                                    \
\ CheckValidofSession %^ (sid,read_prikey,ssn_tbl) = (NO_USER, ssn_tbl)";
by(cut_facts_tac ((CheckValidofSession_axdef RS DECL_D2)::prems) 1);
by (REPEAT (etac ballE 1));
br trans 1;ba 1;
(* by(res_inst_tac [("ACCESS_TYPE","X")] ACCESS_TYPE_induct 1); *)
by(ALLGOALS(asm_simp_tac (simpset() addsimps [image_def, Ball_def,asSet_def])));
by(stac if_P 1); ba 1;
auto();
qed "CheckValidofSession_F2";
(* trick proof synthesizing premise containing Z - projection:
   ! x. x ~: dom ssn_tbl \/
           (sid ~: dom (ssn_tbl %^ x) \/
            accept_read_prikey ~= (ssn_tbl %^ x %^ sid).pkra)
 *)
Addsimps[CheckValidofSession_F2]; (* probably not too useful ... *)


val prems = goal SessionManager.thy
"!!sid. \
\[| sid : dom (gen_un (ran ssn_tbl)); ! x. x ~: dom ssn_tbl | ?X sid x;\
\   ssn_tbl : SESSION_TABLE |] ==>                                    \
\ CheckValidofSession %^ (sid,write_siglog,ssn_tbl) = (NO_USER, ssn_tbl)";
by(cut_facts_tac ((CheckValidofSession_axdef RS DECL_D2)::prems) 1);
by (REPEAT (etac ballE 1));
br trans 1;ba 1;
(* by(res_inst_tac [("ACCESS_TYPE","X")] ACCESS_TYPE_induct 1); *)
by(ALLGOALS(asm_simp_tac (simpset() addsimps [image_def, Ball_def,asSet_def])));
by(stac if_P 1); ba 1;
auto();
qed "CheckValidofSession_F3";
Addsimps[CheckValidofSession_F3]; (* probably not too useful ... *)

val prems = goal SessionManager.thy
"!!sid. \
\[| sid : dom (gen_un (ran ssn_tbl)); ! x. x ~: dom ssn_tbl | ?X sid x;\
\   ssn_tbl : SESSION_TABLE |] ==>                                    \
\ CheckValidofSession %^ (sid,read_siglog,ssn_tbl) = (NO_USER, ssn_tbl)";
by(cut_facts_tac ((CheckValidofSession_axdef RS DECL_D2)::prems) 1);
by (REPEAT (etac ballE 1));
br trans 1;ba 1;
(* by(res_inst_tac [("ACCESS_TYPE","X")] ACCESS_TYPE_induct 1); *)
by(ALLGOALS(asm_simp_tac (simpset() addsimps [image_def, Ball_def,asSet_def])));
by(stac if_P 1); ba 1;
auto();
qed "CheckValidofSession_F4";
Addsimps[CheckValidofSession_F4]; (* probably not too useful ... *)



val prems = goal SessionManager.thy
"[| sid ~: dom (gen_un (ran ssn_tbl)); ssn_tbl : SESSION_TABLE |] ==>   \
\  (sid,acc_typ,ssn_tbl)~: isValidSession_";
by(cut_facts_tac ((isValidSession__axdef RS DECL_D2)::prems) 1);
by(REPEAT (etac ballE 1));
by(res_inst_tac[("P","Not")] ssubst 1); ba 1;
auto();
```

```
qed"isValidSession_1";
Addsimps[isValidSession_1];

val prems = goal SessionManager.thy
"!!sid. [|  (sid,acc_typ,ssn_tbl): isValidSession_;                      \
\           ssn_tbl : SESSION_TABLE |] ==>   \
\       sid : dom (gen_un (ran ssn_tbl))";
be contrapos2 1;
br isValidSession_1 1;
auto();
qed"isValidSession_1'";


val prems = goal SessionManager.thy
"!!sid.                                                                  \
\[| sid : dom (gen_un (ran ssn_tbl));                                    \
\   ! x. x ~: dom ssn_tbl | (sid ~: dom (ssn_tbl %^ x) |                 \
\            accept_read_prikey ~= PROJ (ssn_tbl %^ x %^ sid) fst ''pkra'');   \
\   ssn_tbl : SESSION_TABLE |]                                           \
\  ==> (sid,read_prikey,ssn_tbl)~: isValidSession_";
by(cut_facts_tac ((isValidSession__axdef RS DECL_D2)::prems) 1);
by(REPEAT (etac ballE 1));
by(res_inst_tac[("P","Not")] ssubst 1); ba 1;
auto();
qed"isValidSession_2";
Addsimps[isValidSession_2];
(* abstracts CheckValidofSession_F2 *)


val prems = goal SessionManager.thy
"!!sid.                                                                  \
\[| (sid,read_prikey,ssn_tbl) : isValidSession_;                         \
\   ssn_tbl : SESSION_TABLE |]                                           \
\  ==>  (? x. x : dom ssn_tbl & (sid : dom (ssn_tbl %^ x) &              \
\            accept_read_prikey = PROJ (ssn_tbl %^ x %^ sid) fst ''pkra''))    \
\       & sid : dom (gen_un (ran ssn_tbl))";
by(forward_tac [isValidSession_1'] 1); ba 1;
br conjI 1;
be contrapos2 1;
br isValidSession_2 1;
auto();
qed"isValidSession_2'";


val prems = goal SessionManager.thy
"!!sid.                                                                  \
\[| sid : dom (gen_un (ran ssn_tbl));                                    \
\   ! x. x ~: dom ssn_tbl | (sid ~: dom (ssn_tbl %^ x) |                 \
\            accept_write_siglog ~= PROJ (ssn_tbl %^ x %^ sid) snd ''slwa'');  \
\   ssn_tbl : SESSION_TABLE |]                                           \
\  ==> (sid,write_siglog,ssn_tbl)~: isValidSession_";
by(cut_facts_tac ((isValidSession__axdef RS DECL_D2)::prems) 1);
by(REPEAT (etac ballE 1));
by(res_inst_tac[("P","Not")] ssubst 1); ba 1;
auto();
qed"isValidSession_3";
Addsimps[isValidSession_3];
(* abstracts CheckValidofSession_F3 *)


val prems = goal SessionManager.thy
"!!sid.                                                                  \
```

```
\[| (sid,write_siglog ,ssn_tbl) : isValidSession_;                                \
\   ssn_tbl : SESSION_TABLE |]                                                    \
\  ==>  (? x. x : dom ssn_tbl & (sid : dom (ssn_tbl %^ x) &                       \
\             accept_write_siglog = PROJ (ssn_tbl %^ x %^ sid) snd ''slwa''))    \
\       & sid : dom (gen_un (ran ssn_tbl))";
by(forward_tac [isValidSession_1'] 1); ba 1;
br conjI 1;
be contrapos2 1;
br isValidSession_3 1;
auto();
qed"isValidSession_3'";


(* do not unfold any longer, just use derived rules *)
Delsimps[stripS(isValidSession__axdef RS DECL_D2)];



(* more special attempt *)
val prems = goal SessionManager.thy
"!!sid.                                                                          \
\[| (sid,read_prikey,ssn_tbl): isValidSession_ ; ssn_tbl : SESSION_TABLE   \
\|] ==>   fst(CheckValidofSession %^ (sid, read_prikey, ssn_tbl)) : dom ssn_tbl";
by(stac (stripS(CheckValidofSession_axdef RS DECL_D2)) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps [image_def,Let_def])));
by(case_tac "sid : dom (gen_un (ran ssn_tbl))" 1);
bd isValidSession_1 2;
by(ALLGOALS(Asm_full_simp_tac));
by(Step_tac 2);
by(Step_tac 2);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                               [asSet_def, choose_in_subset,aux1,aux2,aux3])));
by(case_tac "?X" 1);
br if_general_I1 1;ba 1;
be if_general_I2 2;
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                               [asSet_def, choose_in_subset,aux1,aux2,aux3])));
be contrapos2 1;
br isValidSession_2 1;
auto();
qed "CheckValidofSession_uid_in_dom_ssn_tbl1";

val prems = goal SessionManager.thy
"!!sid.                                                                          \
\[| (sid,write_siglog,ssn_tbl): isValidSession_ ; ssn_tbl : SESSION_TABLE  \
\|] ==>   fst(CheckValidofSession %^ (sid, write_siglog, ssn_tbl)) : dom ssn_tbl";
by(stac (stripS(CheckValidofSession_axdef RS DECL_D2)) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps [image_def,Let_def])));
by(case_tac "sid : dom (gen_un (ran ssn_tbl))" 1);
bd isValidSession_1 2;
by(ALLGOALS(Asm_full_simp_tac));
by(Step_tac 2);
by(Step_tac 2);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                               [asSet_def, choose_in_subset,aux1,aux2,aux3])));
by(case_tac "?X" 1);
br if_general_I1 1;ba 1;
be if_general_I2 2;
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                               [asSet_def, choose_in_subset,aux1,aux2,aux3])));
be contrapos2 1;
br isValidSession_3 1;
```

```
auto();
qed "CheckValidofSession_uid_in_dom_ssn_tbl2";


val prems = goal SessionManager.thy
"!!sid.                                                                    \
\[| (sid,read_siglog,ssn_tbl): isValidSession_ ; ssn_tbl : SESSION_TABLE   \
\|] ==>   fst(CheckValidofSession %^ (sid, read_siglog, ssn_tbl)) : dom ssn_tbl";
by(stac (stripS(CheckValidofSession_axdef RS DECL_D2)) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps [image_def,Let_def])));
by(case_tac "sid : dom (gen_un (ran ssn_tbl))" 1);
bd isValidSession_1 2;
by(ALLGOALS(Asm_full_simp_tac));
by(Step_tac 2);
by(Step_tac 2);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                               [asSet_def, choose_in_subset,aux1,aux2,aux3])));
qed "CheckValidofSession_uid_in_dom_ssn_tbl3";


(* premise can be weakened to sid : dom (gen_un (ran ssn_tbl)) *)
(* a nice little tricky lemma ... and a vital invariant ... *)
val [valid_session,ssn_c, uid_c, invert] = goal SessionManager.thy
"[| (sid, read_prikey, ssn_tbl) : isValidSession_;                          \
\   ssn_tbl : SESSION_TABLE; uid : dom ssn_tbl;                             \
\   ! x: dom ssn_tbl. ! y: dom ssn_tbl.                                     \
\        (? s. s:dom(ssn_tbl %^ x) & s:dom(ssn_tbl %^ y))  =+=> (x = y)|]   \
\ ==> (sid : dom(snd(CheckValidofSession %^ (sid,read_prikey,ssn_tbl)) %^ uid))   \
\     = (sid : dom (ssn_tbl %^ uid))";
by(cut_facts_tac [ssn_c,ssn_c RS (valid_session RS isValidSession_2')] 1);
be conjE 1; be exE 1;
by(REPEAT(etac conjE 1 ORELSE etac exE 1));
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_simp_tac (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(stac if_not_P 1);
by(Blast_tac 1);
by(res_inst_tac [("s","x"),("t","SessionManager.choose %^ ?X")] subst 1);
by(ALLGOALS(Asm_simp_tac));
br (invert RS (ssn_c RS (choose_unique RS sym))) 1; ba 1; ba 1;
by(case_tac "sid : dom (ssn_tbl %^ uid)" 1);
by(ALLGOALS(Asm_simp_tac));
by(HINT " x = uid" (K all_tac) 1);
by(HINT " uid ~= x" (K all_tac) 3);
by(ALLGOALS(Asm_simp_tac));
be (stripS invert) 1;
br uid_c 1;
by(Blast_tac 1);
be swap 1;
by(Asm_full_simp_tac 1);
qed "sid_ind_dom_CheckValidofSession_inv";



val [quod, valid_session,ssn, uid_c, invert] = goal SessionManager.thy
"[|  s : dom (snd (CheckValidofSession %^ (sid,read_prikey, ssn_tbl)) %^ x);      \
\    (sid, read_prikey, ssn_tbl) : isValidSession_;                             \
\    ssn_tbl : SESSION_TABLE;  x : dom ssn_tbl;                                 \
\    ! x: dom ssn_tbl. ! y: dom ssn_tbl.                                        \
\        (? s. s:dom(ssn_tbl %^ x) & s:dom(ssn_tbl %^ y))  =+=> (x = y)         \
\ |] ==> s : dom (ssn_tbl %^ x)";
by(cut_facts_tac [quod,ssn,ssn RS (valid_session RS isValidSession_2')] 1);
be conjE 1; be exE 1;
```

```
by(REPEAT(etac conjE 1 ORELSE etac exE 1));
be contrapos2 1;
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_simp_tac (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(stac if_not_P 1);
by(Blast_tac 1);
by(thin_tac "accept_read_prikey = ?X" 1);
by(res_inst_tac
   [("s","xa"),
    ("t","SessionManager.choose %^ {y. y:dom ssn_tbl & sid:dom(ssn_tbl %^ y)}")] ssubst 1);
be (invert RS (ssn RS choose_unique)) 1; ba 1;
by(case_tac "x =xa" 1);
by(ALLGOALS(Asm_simp_tac));
be swap 1;
by(ALLGOALS(Asm_full_simp_tac));
qed"sid_in_Check_implies_sid_in_ssn_tbl";



val [valid_session,ssn_c, uid_c, invert] = goal SessionManager.thy
"[| (sid, write_siglog, ssn_tbl) : isValidSession_;
\
\   ssn_tbl : SESSION_TABLE; uid : dom ssn_tbl;                                    \
\   ! x: dom ssn_tbl. ! y: dom ssn_tbl.                                            \
\        (? s. s:dom(ssn_tbl %^ x) & s:dom(ssn_tbl %^ y))  =+=> (x = y)|]          \
\ ==> (sid : dom(snd(CheckValidofSession %^ (sid,write_siglog,ssn_tbl)) %^ uid))
\
\     = (sid : dom (ssn_tbl %^ uid))";
by(cut_facts_tac [ssn_c,ssn_c RS (valid_session RS isValidSession_3')] 1);
be conjE 1; be exE 1;
by(REPEAT(etac conjE 1 ORELSE etac exE 1));
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_simp_tac (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(stac if_not_P 1);
by(Blast_tac 1);
by(res_inst_tac [("s","x"),("t","SessionManager.choose %^ ?X")] subst 1);
by(ALLGOALS(Asm_simp_tac));
br (invert RS (ssn_c RS (choose_unique RS sym))) 1; ba 1; ba 1;
by(case_tac "sid : dom (ssn_tbl %^ uid)" 1);
by(ALLGOALS(Asm_simp_tac));
by(HINT " x = uid" (K all_tac) 1);
by(HINT " uid ~= x" (K all_tac) 3);
by(ALLGOALS(Asm_simp_tac));
be (stripS invert) 1;
br uid_c 1;
by(Blast_tac 1);
be swap 1;
by(Asm_full_simp_tac 1);
qed "sid_ind_dom_CheckValidofSession_inv2";


(* ********************************************************************** *)
(*       Project          : HSD security analysis
         Author           : B. Wolff
         Affiliation      : ETH Z\"urich
         This theory      : State and Operation Schemas of HSD
                            (Version based on Latex-ZETA format)
         $Date: 2002/12/11 14:35:46 $
         $Revision: 1.3 $
         Release          : 2.5                                          *)
(* ********************************************************************** *)

(* cd "holz";
```

```
   use_holz "HSD";
 *)

toToplevel HSD.axdefs;
toToplevel HSD.schemes;

Add_axdefs_TC (map snd HSD.axdefs);

(* Windows Side Operations - nothing to do wrt. state invariance lemmas,
   since windows side has no own state. *)

zgoal HSD.thy
"AuthenticateUserL =+=> (signature_log' =  signature_log &            \
\                        access_control_list' = access_control_list & \
\                        pri_key_list' = pri_key_list)";
by(stripS_tac 1);
by(forward_tac [stripS (get_decl HSD.thy "AuthenticateUserL" 2)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AuthenticateUserL" 3)] 1);
by (convert2hol_tac [] 1);
qed"AuthenticateUserL_inv_state_components";


zgoal HSD.thy
"AuthenticateUserL                                                  \
\  =+=>  uid : dom session_table                                    \
\  =+=>  (session_table' %^ uid = session_table %^ uid)";
by(stripS_tac 1);
by(zftac (get_conj HSD.thy "AuthenticateUserL" 2) 1);
by(zftac (get_conj HSD.thy "AuthenticateUserL" 3) 1);
by(thin_tac "Authentication = ?X" 1);
by(thin_tac "session_table' = ?X" 1);
(* saturation of tc's *)
by(zftac (get_decl HSD.thy "AuthenticateUserL" 1) 1);
by(zftac (get_decl HSD.thy "AuthenticateUserL" 3) 1);
by(zftac (get_decl HSD.thy "AuthenticateUserL" 7) 1);
by(zdtac (get_decl HSD.thy "DARMA" 4) 1);
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
by(zdtac (get_decl HSD.thy "SessionManager" 2) 1);
by(zdtac (get_decl HSD.thy "AccessController" 1) 1);
(* <<< saturation *)
by(zstac (AuthenticateUser_axdef RS DECL_D2) 1);
by(zstac (RegistSessionInformation_axdef RS DECL_D2) 1);
by(case_tac "User_authentication_uid = uid" 1);
by(ALLGOALS(simp_tac (simpset() addsimps  [maplet_def]
                                addsplits [expand_if])));
bd (neq_sym RS iffD1) 2;
by(ALLGOALS(Asm_simp_tac));
qed"AuthenticateUserL_uid_auth_implies_session_table_inv";



zgoal HSD.thy
"GenerateSignatureL =+=> (access_control_list' = access_control_list & \
\                         session_IDs = session_IDs'  &                \
\                         pri_key_list' = pri_key_list)";
by(stripS_tac 1);
by(forward_tac [stripS (get_decl HSD.thy "GenerateSignatureL" 3)] 1);
by(dtac (stripS(get_conj HSD.thy "GenerateSignatureL" 6)) 1);
by (convert2hol_tac [] 1);
```

```
qed"GenerateSignatureL_inv_state_components";


Delsimps[No_Dom_Restr];

(* <<<<<<<<<<<<<<<<<<<<<<< *)

zgoal HSD.thy
"GenerateSignatureL =+=> dom(signature_log) <= dom(signature_log')";
by(stripS_tac 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
auto();
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(forward_tac [stripS (get_decl HSD.thy "HysteresisSignature" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 2)] 1);
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;

by(forw_inst_tac [("f","% x. fst(snd x)")] arg_cong 1);by(rotate_tac ~1 1);
by(forw_inst_tac [("f","% x. fst(snd x)")] arg_cong 2);by(rotate_tac ~1 2);
bd pair_rel_dom_fst 1;
by(ALLGOALS(asm_full_simp_tac (prod_ss addsimps [Let_def])));

by(REPEAT (etac conjE 1));
by(stac (stripS(AppendSignatureRecord_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
br impI 1;
br (CheckValidofSession_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(ALLGOALS(Asm_simp_tac));

by(asm_simp_tac (prod_ss addsimps [Let_def] addsplits [expand_if]) 1);
auto();
qed"GenerateSignatureL_siglog_mono";


zgoal HSD.thy
"GenerateSignatureL =+=>                                              \
\ Signature_generation_sid ~: dom (gen_un (ran session_table)) =+=> \
\ signature_log' = signature_log & session_table' = session_table";
by(stripS_tac 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
be DECL_E 1;
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(forward_tac [stripS (get_decl HSD.thy "HysteresisSignature" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 2)] 1);
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;
by(Asm_full_simp_tac 1);
by(HINT "(Signature_generation_sid, session_table, \
        \ pri_key_list) : ReadPrivateKeyFailure_" (K all_tac) 1);
by(rotate_tac ~1 1);
by(asm_full_simp_tac HOL_ss 1);
br (isValidSessionVSReadPrivateKeyFailure1 RS mp) 1;
by(ALLGOALS(asm_full_simp_tac HOL_ss));
be isValidSession_1 1;ba 1;
```

```
qed"GenerateSignatureL_inv_if_invalid_session";


zgoal HSD.thy
"GenerateSignatureL =+=>                                \
\   (dom session_table <= dom pri_key_list &            \
\    dom signature_log <= dom pri_key_list) =+=>        \
\   (dom session_table' <= dom pri_key_list' &          \
\    dom signature_log' <= dom pri_key_list')";
by(stripS_tac 1);
by(forward_tac [stripS GenerateSignatureL_inv_state_components] 1);
by(forward_tac [stripS (get_decl HSD.thy "GenerateSignatureL" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "GenerateSignatureL" 2)] 1);
by(forward_tac [stripS (get_decl HSD.thy "GenerateSignatureL" 3)] 1);
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(case_tac "Signature_generation_sid : dom (gen_un (ran session_table))" 1);
bd(stripS GenerateSignatureL_inv_if_invalid_session) 2;
auto();
by(dtac (stripS(get_conj HSD.thy "GenerateSignatureL" 2)) 1);
be if_eqL_E 1;
by(forw_inst_tac [("f","% x. snd(snd x)")] arg_cong 1);by(rotate_tac ~1 1);
by(forw_inst_tac [("f","% x. snd(snd x)")] arg_cong 2);by(rotate_tac ~1 2);
by(ALLGOALS(asm_full_simp_tac (prod_ss addsimps [Let_def])));
auto();
by(thin_tac "?X" 1);
by(eres_inst_tac [("Q","(xa,ya) : ?Y")] contrapos2 1);
by(stac (stripS(AppendSignatureRecord_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
br(ReadPrivateKey_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(Asm_simp_tac 1);
by(dtac (stripS (get_decl HSD.thy "SessionManager" 1)) 1);
by(dtac (stripS (get_decl HSD.thy "AccessController" 2)) 1);
by(dtac (stripS (get_decl HSD.thy "HysteresisSignature" 1)) 2);
by(ALLGOALS Asm_full_simp_tac);
by(asm_simp_tac (simpset() addsimps [Let_def,maplet_def] addsplits [expand_if]) 1);
br conjI 1; by(strip_tac 1);by(strip_tac 2);
auto();

by(dres_inst_tac [("a","(xa, ya)")] pair_rel_dom_fst 1);
by(Asm_full_simp_tac 1);
by(dtac (stripS (get_decl HSD.thy "SessionManager" 1)) 1);
by(dtac (stripS (get_decl HSD.thy "AccessController" 2)) 1);
by(rotate_tac ~2 1);
by(asm_full_simp_tac (HOL_ss addsimps [ReadPrivateKey_dom_session_table_inv]) 1);
auto();

by(dres_inst_tac [("a","(xa, ya)")] pair_rel_dom_fst 1);
by(Asm_full_simp_tac 1);
by(dtac (stripS (get_decl HSD.thy "SessionManager" 1)) 1);
by(dtac (stripS (get_decl HSD.thy "AccessController" 2)) 1);
by(rotate_tac ~2 1);
by(eres_inst_tac [("Q","xa : ?X")] contrapos2 1);
by(stac CheckValidofSession_dom_session_table_inv 1);
br (ReadPrivateKey_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(Asm_simp_tac 1); ba 1;
by(stac ReadPrivateKey_dom_session_table_inv 1);
by(ALLGOALS(Asm_simp_tac));
auto();

by(dtac (stripS(get_conj HSD.thy "GenerateSignatureL" 2)) 1);
be if_eqL_E 1;
```

```
by(forw_inst_tac [("f","% x. snd(snd x)")] arg_cong 1);by(rotate_tac ~1 1);
by(forw_inst_tac [("f","% x. snd(snd x)")] arg_cong 2);by(rotate_tac ~1 2);
by(ALLGOALS(asm_full_simp_tac (prod_ss addsimps [Let_def])));
auto();

by(forw_inst_tac [("f","fst")] arg_cong 1);back();
by(rotate_tac ~1 1);
by(asm_full_simp_tac prod_ss 1);
by(thin_tac "(fst ?X,snd ?Y) = ?Z" 1);
by(dres_inst_tac [("a","(xa, ya)")] pair_rel_dom_fst 1);
by(Asm_full_simp_tac 1);
by(eres_inst_tac [("Q","xa : ?X")] contrapos2 1);
by(dtac (stripS (get_decl HSD.thy "SessionManager" 1)) 1);
by(dtac (stripS (get_decl HSD.thy "AccessController" 2)) 1);
by(dtac (stripS (get_decl HSD.thy "HysteresisSignature" 1)) 1);
by(stac (stripS(AppendSignatureRecord_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
br impI 1;
br (CheckValidofSession_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(ALLGOALS(Asm_simp_tac));
by(simp_tac (prod_ss addsimps [Let_def,maplet_def] addsplits [expand_if]) 1);
auto();

by(eres_inst_tac [("Pa","fst(?X) : dom pri_key_list")] swap 1);
bd CheckValidofSession_uid_in_dom_ssn_tbl2 1; ba 1;
auto();

by(eres_inst_tac [("Pa","fst(?X) : dom pri_key_list")] swap 1);
bd CheckValidofSession_uid_in_dom_ssn_tbl2 1;
br (CheckValidofSession_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(ALLGOALS(Asm_simp_tac));
be subsetD 1;

by(asm_full_simp_tac (simpset() ) 1);
qed"GenerateSignatureL_implies_pri_key_list_bounds";


zgoal HSD.thy
" GenerateSignatureL =+=> uid ~: dom session_table =+=>             \
\ ( uid : dom signature_log &                                       \
\   signature_log %^ uid = signature_log' %^ uid) |                 \
\ ( uid ~: dom signature_log & uid ~: dom signature_log')";
by(stripS_tac 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
be DECL_E 1;
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(forward_tac [stripS (get_decl HSD.thy "HysteresisSignature" 1)] 1);
(*by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1); *)
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 2)] 1);
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;
by(ALLGOALS(Asm_full_simp_tac));
by(REPEAT (etac conjE 1));
by(thin_tac "Signature = ?X" 1);
by(dres_inst_tac [("f","fst")]arg_cong 1);back();by(rotate_tac ~1 1);
by(ALLGOALS(Asm_full_simp_tac));
by(forward_tac [not_ReadPrivateKeyFailure1VSisValidSession] 1);
by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 2);
by(ALLGOALS(Asm_full_simp_tac));
```

```
by(thin_tac "Command = ?X" 1);
by(thin_tac "SNAME AccessController ?X" 1);
by(thin_tac "SNAME HysteresisSignature ?X" 1);
by(forward_tac [CheckValidofSession_uid_in_dom_ssn_tbl1] 1); (* nec ? *)
be (stripS (get_decl HSD.thy "SessionManager" 1)) 1;

by(case_tac "uid : dom signature_log" 1);
by(ALLGOALS(Asm_simp_tac));
by(stac (stripS(AppendSignatureRecord_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));

br(ReadPrivateKey_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(Asm_simp_tac 1);
be (stripS (get_decl HSD.thy "SessionManager" 1)) 1;
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
br impI 1;
by(stac oplus_apply2 1); br refl 2;
bd CheckValidofSession_uid_in_dom_ssn_tbl2 1;
br(ReadPrivateKey_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(Asm_simp_tac 1);
be (stripS (get_decl HSD.thy "SessionManager" 1)) 1;

by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1);
by(rotate_tac ~1 1);
by(asm_full_simp_tac (simpset() addsimps [Let_def,maplet_def]) 1);
by(Blast_tac 1);

by(stac (stripS(AppendSignatureRecord_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
br(ReadPrivateKey_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(Asm_simp_tac 1);
be (stripS (get_decl HSD.thy "SessionManager" 1)) 1;
by(asm_full_simp_tac (simpset() addsimps [Let_def,maplet_def]) 1);

by(asm_full_simp_tac (simpset() addsimps [Let_def,maplet_def]
                                 addsplits [expand_if]) 1);
br impI 1;
bd CheckValidofSession_uid_in_dom_ssn_tbl2 1;
br(ReadPrivateKey_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(Asm_simp_tac 1);
be (stripS (get_decl HSD.thy "SessionManager" 1)) 1;
by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1);
by(rotate_tac ~1 1);
by(asm_full_simp_tac (simpset() addsimps [Let_def,maplet_def]) 1);
by(Blast_tac 1);
qed"GenerateSignatureL_implies_not_siglogChanges";
(* This theorem is hard !!! *)



(* version pre facto *)
zgoal HSD.thy
"GenerateSignatureL =+=>                                                    \
\ ((uid : dom signature_log' /\\ uid ~: dom signature_log) \\/               \
\  (uid : dom signature_log  /\\ signature_log %^ uid ~= signature_log' %^ uid)) =+=> \
\  uid : dom session_table /\\ Signature_generation_sid : dom (session_table %^ uid)";
by(stripS_tac 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
be DECL_E 1;
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
(* sig saturation *)
```

```
by(zftac (get_decl HSD.thy "HysteresisSignature" 1) 1);
(*by(zftac (get_decl HSD.thy "SessionManager" 1) 1); *)
by(zftac (get_decl HSD.thy "AccessController" 1) 1);
by(zftac (get_decl HSD.thy "AccessController" 2) 1);
(* <<< sig saturation *)
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;
by(ALLGOALS(Asm_full_simp_tac));
by(REPEAT (etac conjE 1));
by(Blast_tac 1);
by(asm_full_simp_tac (simpset() addsimps [Let_def])1);
by(REPEAT (etac conjE 1));
by (zftac (get_decl HSD.thy "SessionManager" 1) 1);
by(HINT "SESSION_ID = UNIV" (fn _ => (rtac set_ext 2) THEN (Asm_simp_tac 2)) 1);
by (zftac (not_ReadPrivateKeyFailure1VSisValidSession) 1);
by (zftac (AppendSignatureRecordFailure_VS_isValidSession RS mp) 1);
by(ALLGOALS(Asm_full_simp_tac));
(* exploit in invertibility of session_table *)
br ballI 1; br ballI 1; br impI 1;
be (stripS(get_conj HSD.thy "SessionManager" 1)) 1;
by(ALLGOALS(Asm_simp_tac));
(* <<< exploit ... *)
by(zftac AppendSignatureRecord_lemma2 1);
by(dres_inst_tac [("f","fst")] arg_cong 3);back();
by(Asm_full_simp_tac 3);
by(ALLGOALS tc_tac);
by(zstac (ReadPrivateKey_axdef RS DECL_D2) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps [Let_def])));
(* exploit in invertibility of session_table *)
br ballI 1; br ballI 1; br impI 1;
be (stripS(get_conj HSD.thy "SessionManager" 1)) 1;
by(ALLGOALS(Asm_simp_tac));
(* <<< exploit ... *)
by(HINT "uid : dom session_table" (K all_tac) 1);
by(res_inst_tac [("t","uid")] subst 2); ba 2;
br CheckValidofSession_uid_in_dom_ssn_tbl1 2;
by(ALLGOALS(Asm_simp_tac));
by(zftac isValidSession_2' 1);
be conjE 1; be exE 1;by(REPEAT (etac conjE 1));
by(HINT "uid = x" (K all_tac) 1);
by(ALLGOALS(Asm_simp_tac));
(* <<< exploit ... *)
be (stripS(get_conj HSD.thy "SessionManager" 1)) 1;
by(ALLGOALS(Asm_simp_tac));
br exI 1; br conjI 1; ba 2;
by(eres_inst_tac [("Q","?X = uid")] contrapos2 1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_simp_tac (simpset()
                 addsimps  [asSet_def,image_def,Let_def,maplet_def]
                 addsplits [expand_if] ) 1);
by(HINT "NO_USER ~= uid"  (fn _ => (dtac NO_USER_not_in_dom_SESSION_TABLE 2)
                              THEN (Blast_tac 2))   1);
by(Asm_simp_tac 1);
br impI 1;
br choose_neq_X 1;
by(Blast_tac 3);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                                [asSet_def,choose_in_subset,aux1,aux2,aux2',aux3])));
qed"GenerateSignatureL_siglogChanges_charn";
```

```
(* version post facto *)
(* proof style: experimental. without tc-saturation *)
zgoal HSD.thy
"GenerateSignatureL =+=>                                                    \
\ ((uid : dom signature_log' /\\ uid ~: dom signature_log) \\/              \
\  (uid : dom signature_log  /\\ signature_log %^ uid ~= signature_log' %^ uid)) =+=> \
\  uid : dom session_table' /\\ Signature_generation_sid : dom (session_table' %^ uid)";
by(stripS_tac 1);
by(zftac (GenerateSignatureL_siglogChanges_charn) 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
be DECL_E 1;
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;
by(ALLGOALS(Asm_full_simp_tac));
by(REPEAT (etac conjE 1));
by(thin_tac "Signature = ?X" 1);
by(dres_inst_tac [("f","snd")] arg_cong 1);back();
by(ALLGOALS(Asm_full_simp_tac));
by(zstac (AppendSignatureRecord_axdef RS DECL_D2) 1);
  by(zetac (get_decl HSD.thy "HysteresisSignature" 1) 2);
  by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);
  by(tc_tac 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps  [asSet_def,image_def,Let_def,maplet_def]
                                    addsplits [expand_if])));
by(zstac (stripS(ReadPrivateKey_axdef RS DECL_D2)) 1);
  by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);
  by (zftac (not_ReadPrivateKeyFailure1VSisValidSession) 1);
  by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps  [asSet_def,image_def,Let_def,maplet_def]
                                    addsplits [expand_if])));
by (zstac CheckValidofSession_dom_session_table_inv 1);
by (zstac CheckValidofSession_dom_session_table_inv 3);
by(Asm_simp_tac 5);
  by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);
  by(tc_tac 1);
  by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);

by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
by(zdtac (AppendSignatureRecordFailure_VS_isValidSession RS mp) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);
  by(zetac (get_decl HSD.thy "HysteresisSignature" 1) 2);

by(stripS_tac 1);
by(HINT "SESSION_ID = UNIV" (fn _ => (rtac set_ext 2) THEN (Asm_simp_tac 2)) 1);
be (stripS(get_conj HSD.thy "SessionManager" 1)) 1;
by(ALLGOALS(Asm_simp_tac));

by(zstac sid_ind_dom_CheckValidofSession_inv2 1);
by(zstac sid_ind_dom_CheckValidofSession_inv 4);
by(ALLGOALS(Asm_simp_tac));
by(stripS_tac 5);
by(HINT "SESSION_ID = UNIV" (fn _ => (rtac set_ext 6) THEN (Asm_simp_tac 6)) 5);
by(zetac (get_decl HSD.thy "SessionManager" 1) 4);
by(zrtac (get_conj HSD.thy "SessionManager" 1) 4);
by(ALLGOALS(Asm_simp_tac));
```

```
  by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);
by(ALLGOALS(Asm_simp_tac));


  by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);
by(ALLGOALS(Asm_simp_tac));

  by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
  by(zftac (get_decl HSD.thy "AccessController" 2) 1);
by(stripS_tac 1);
br (stripS(get_conj HSD.thy "SessionManager" 1)) 1; ba 1;
by(HINT "SESSION_ID = UNIV" (fn _ => (rtac set_ext 4) THEN (Asm_simp_tac 4)) 3);
by(ALLGOALS(Asm_full_simp_tac));
by(thin_tac "session_table' = ?X" 1);
by(thin_tac "SNAME DARMA ?X" 1);
by(thin_tac "?X | ?Y" 1);
be exE 1;
by(REPEAT(etac conjE 1));
bd sid_in_Check_implies_sid_in_ssn_tbl 1;
bd sid_in_Check_implies_sid_in_ssn_tbl 5;
auto();
by(zrtac (get_conj HSD.thy "SessionManager" 1) 2);
by(zrtac (get_conj HSD.thy "SessionManager" 1) 1);
auto();
qed"GenerateSignatureL_siglogChanges_charn2";



zgoal HSD.thy
"GenerateSignatureL =+=>                                              \
\   (signature_log %^ uid ~= signature_log' %^ uid |                   \
\    uid ~: dom signature_log & uid : dom signature_log') =+=>         \
\   (! uid':dom signature_log - {uid}.                                 \
\                    signature_log %^ uid' = signature_log' %^ uid')";
by(stripS_tac 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
be DECL_E 1;
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(forward_tac [stripS (get_decl HSD.thy "HysteresisSignature" 1)] 1);
(*by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1); *)
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 2)] 1);
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;
by(ALLGOALS(Asm_full_simp_tac));
by(REPEAT (etac conjE 1));
by(dres_inst_tac [("f","fst")] arg_cong 1);back();by(rotate_tac ~1 1);
by(ALLGOALS(Asm_full_simp_tac));
by(forward_tac [not_ReadPrivateKeyFailure1VSisValidSession] 1);
by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 2);
by(ALLGOALS(Asm_full_simp_tac));
by(stac (stripS (AppendSignatureRecord_axdef RS DECL_D2)) 1);
by(ALLGOALS(Asm_simp_tac));
br(ReadPrivateKey_axdef RS DECL_D1 RS tfun_apply_snd) 1;
by(Asm_simp_tac 1);
be (stripS (get_decl HSD.thy "SessionManager" 1)) 1;
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
br impI 1;
```

```
by(simp_tac (simpset() addsimps [maplet_def]) 1);
by(stac oplus_non_apply 1);
br refl 2;
by(thin_tac "?X" 1);
be disjE 1;
be swap 1;
by(Asm_full_simp_tac 1);
by(thin_tac "uid' = ?X" 1);
by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1);
auto();
by(ALLGOALS(asm_simp_tac (simpset() addsimps [Let_def])));
by(eres_inst_tac [("Pa","?X = ?Y")] swap 1);back();
by(stac (stripS (ReadPrivateKey_axdef RS DECL_D2)) 2);
by(stac (stripS (ReadPrivateKey_axdef RS DECL_D2)) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps [Let_def])));
be (stripS (get_decl HSD.thy "SessionManager" 1)) 3;
be (stripS (get_decl HSD.thy "SessionManager" 1)) 1;
br AppendSignatureRecord_lemma1 1;
by(ALLGOALS(Asm_simp_tac));
be (stripS (get_decl HSD.thy "SessionManager" 1)) 2;
br disjI2 1;
auto();
by(eres_inst_tac [("Pa","?X = ?Y")] swap 1);
br AppendSignatureRecord_lemma1 1;
by(ALLGOALS(Asm_simp_tac));
be (stripS (get_decl HSD.thy "SessionManager" 1)) 2;
br disjI1 1;
by(dres_inst_tac [("a","(uid, ya)")] pair_rel_dom_fst 1);
by(ALLGOALS(Asm_full_simp_tac));
qed"GenerateSignatureL_and_siglogChanges_implies_inv_others";



zgoal HSD.thy
"GenerateSignatureL =+=>                                                    \
\ ((uid : dom signature_log' /\\ uid ~: dom signature_log) \\/               \
\  (uid : dom signature_log  /\\ signature_log %^ uid ~= signature_log' %^ uid)) =+=> \
\     (? siglog:SIGNATURE.
\
\        ? hmg:seq CHAR.                                                    \
\            signature_log' %^ uid = hys_sig_gen %^ (hmg, pri_key_list %^ uid, siglog))";
by(stripS_tac 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
be DECL_E 1;
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(forward_tac [stripS (get_decl HSD.thy "HysteresisSignature" 1)] 1);
(*by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1); *)
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 2)] 1);
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;
by(ALLGOALS(Asm_full_simp_tac));
by(REPEAT (etac conjE 1));
by(rotate_tac ~2 1);
by(Asm_full_simp_tac 1);
by(REPEAT (etac conjE 1));
by(dres_inst_tac [("f","fst")] arg_cong 1);back();
by(rotate_tac ~1 1);
by(ALLGOALS(Asm_full_simp_tac));
by (forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1);
```

```
by(forward_tac [not_ReadPrivateKeyFailure1VSisValidSession] 1);
by(forward_tac [AppendSignatureRecordFailure_VS_isValidSession RS mp] 4);
by(ALLGOALS(Asm_full_simp_tac));
(* exploit in invertibility of session_table *)
br ballI 1;
br ballI 1; br impI 1;
be (stripS(get_conj HSD.thy "SessionManager" 1)) 1;
by(HINT "SESSION_ID = UNIV" (K all_tac) 3);
br set_ext 4;
by(ALLGOALS(Asm_simp_tac));

(* unfold AppendSignatureRecord and side conditions *)
by(zstac (AppendSignatureRecord_axdef RS DECL_D2) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps [Let_def,maplet_def])));
by(zrtac (CheckValidofSession_axdef RS DECL_D1 RS tfun_apply_snd) 1);

(* Now reduce the main goal to the basics ... *)
by(stac oplus_by_pair_apply1 1);
by(defer_tac 1); (* push away equality condition *)
br bexI 1; br bexI 1; ba 2;
by(res_inst_tac [("t","uid")] subst 1);
br refl 2; (* thats it ... *)
by(tc_tac 2);


(* Now cleanup: proof applicabilities,
   in particular

  fst (CheckValidofSession %^ (Signature_generation_sid,
                               read_prikey, session_table)) = uid

  and

  fst (CheckValidofSession %^ (Signature_generation_sid,
  write_siglog,
  snd (CheckValidofSession %^ (Signature_generation_sid, read_prikey,
                               session_table)))) = uid *)
br sym 2;
by (zrtac AppendSignatureRecord_lemma1 2);
by (zrtac AppendSignatureRecord_lemma2 1);
by(defer_tac 1);

br ballI 1;
br ballI 1;
br impI 1;
be exE 1;
by (zetac (get_conj HSD.thy "SessionManager" 1) 1);
by(res_inst_tac [("x","s")] bexI 1);
by(Blast_tac 1);
by(tc_tac 1);
by(zstac (ReadPrivateKey_axdef RS DECL_D2) 2);
by(zstac (ReadPrivateKey_axdef RS DECL_D2) 1);
by(ALLGOALS(asm_simp_tac (simpset() addsimps [Let_def])));
qed"GenerateSignatureL_and_siglogChanges_implies_prikey_use";



zgoal HSD.thy
"GenerateSignatureL =+=>                                                    \
\ ((uid : dom signature_log' /\\ uid ~: dom signature_log) \\/               \
\  (uid : dom signature_log  /\\ signature_log %^ uid ~= signature_log' %^ uid)) =+=> \
```

```
\  accept_read_prikey ~= PROJ (session_table' %^ uid %^ Signature_generation_sid) fst ''pkra'''";
by(stripS_tac 1);
by(zftac GenerateSignatureL_siglogChanges_charn 1);
by(HINT "SESSION_ID = UNIV" (fn _ => (rtac set_ext 2) THEN (Asm_simp_tac 2)) 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
be DECL_E 1;
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(forward_tac [stripS (get_decl HSD.thy "HysteresisSignature" 1)] 1);
(*by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1); *)
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 2)] 1);
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;
by(ALLGOALS(Asm_full_simp_tac));
by(REPEAT (etac conjE 1));
by(rotate_tac ~3 1);
by(ALLGOALS(Asm_full_simp_tac));

by(REPEAT (etac conjE 1));
by(dres_inst_tac [("f","snd")] arg_cong 1);back();
by(ALLGOALS(Asm_full_simp_tac));

by(zftac (not_ReadPrivateKeyFailure1VSisValidSession) 1);
by(zetac (get_decl HSD.thy "SessionManager" 1) 1);
by(thin_tac "session_table' = ?X" 1);
by(zftac (get_decl HSD.thy "SessionManager" 1) 1);
by(ALLGOALS(Asm_full_simp_tac));
by(zstac (AppendSignatureRecord_axdef RS DECL_D2) 1);
by(simp_tac (HOL_ss addsimps [Let_def]) 1);
by(tc_tac 1);
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
by(forward_tac [AppendSignatureRecordFailure_VS_isValidSession RS mp] 1);
by(ALLGOALS(Asm_full_simp_tac));
(* exploit in invertibility of session_table *)
br ballI 1; br ballI 1; br impI 1;
be (stripS(get_conj HSD.thy "SessionManager" 1)) 1;
by(ALLGOALS(Asm_simp_tac));
by(zftac isValidSession_2' 1);
by(zftac isValidSession_3' 1);
by(REPEAT(etac conjE 1 ORELSE etac exE 1));
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(simp_tac (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(stac if_P 1);
by(Asm_full_simp_tac 1);
by(stac if_not_P 1);
by(Blast_tac 1);
by(dres_inst_tac [("s","accept_read_prikey")] sym 1);
by(ALLGOALS(Asm_simp_tac));
by(res_inst_tac [("s","uid"),("t","SessionManager.choose %^ ?X")] subst 1);
by(ALLGOALS(Asm_simp_tac));
by(defer_tac 1);
by(zstac (CheckValidofSession_axdef RS DECL_D2) 1);
by(asm_simp_tac (simpset() addsimps [Let_def,asSet_def,image_def,maplet_def])1);
by(stac if_not_P 1);
by(Asm_simp_tac 1);
by(zetac (isValidSession_2' RS conjunct1) 1);
by(res_inst_tac [("s","uid"),("t","SessionManager.choose %^ ?X")] subst 1);
by(zrtac (choose_unique RS sym) 1);
br ballI 1; br ballI 1; br impI 1;
be (stripS(get_conj HSD.thy "SessionManager" 1)) 1;
by(ALLGOALS(Asm_simp_tac));
```

```
by(asm_simp_tac (simpset() addsimps [PROJ_def])1);
br(choose_unique' RS sym) 1;
br set_ext 1;
by(ALLGOALS(Asm_simp_tac));
by(stac (refl RS conj_cong) 1);
be sid_ind_dom_CheckValidofSession_inv 1;
by(ALLGOALS(Asm_simp_tac));
br iffI 2;
by(hyp_subst_tac 3);
by(Asm_simp_tac 3);
be (stripS(get_conj HSD.thy "SessionManager" 1)) 2;
by(ALLGOALS(Asm_simp_tac));
by(res_inst_tac [("x","Signature_generation_sid")] exI 2);
by(Asm_simp_tac 2);
by(stripS_tac 1);
be exE 1;
be (stripS(get_conj HSD.thy "SessionManager" 1)) 1;
by(ALLGOALS(Asm_simp_tac));
by(res_inst_tac [("x","s")] exI 1);
by(ALLGOALS(Asm_simp_tac));
qed"GenerateSignatureL_implies_no_accept_key";



(* The following crucial theorem establishes at the data model level,
   that the operation GenerateSignatureL will never change the
   session_table' and signature_log' *for an authenticated user*,
   if in the session table accept_read_prikey is not set.
   Note that this does not imply that session_table and
   signature_log are unchanged; GenerateSignatureL may process
   another user successfully.
   This is a core proof for HSD_3 on the data level.
 *)
zgoal HSD.thy
"GenerateSignatureL =+=>                                          \
\ (uid : dom session_table /\\                                    \
\  sid : dom (session_table %^ uid) /\\                           \
\  accept_read_prikey ~=                                          \
\   PROJ (session_table %^ uid %^ sid) fst ''pkra'')              \
\ --> (session_table' %^ uid = session_table %^ uid  /\\          \
\      signature_log' %^ uid = signature_log %^ uid)";
by(stripS_tac 1);
by(full_expand_schema_tac GenerateSignatureL_def 1);
be DECL_E 1;
by(asm_full_simp_tac (HOL_ss addsimps [XI_def,DELTA_def]) 1);
by(REPEAT (etac conjE 1));
by(forward_tac [stripS (get_decl HSD.thy "HysteresisSignature" 1)] 1);
(*by(forward_tac [stripS (get_decl HSD.thy "SessionManager" 1)] 1); *)
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "AccessController" 2)] 1);
by(asm_full_simp_tac (HOL_ss addsimps [Let_def])1);
be if_eqL_E 1;
by(ALLGOALS(Asm_full_simp_tac));
by(REPEAT (etac conjE 1));
by(zftac (not_ReadPrivateKeyFailure1VSisValidSession) 1);
by(zetac (get_decl HSD.thy "SessionManager" 1) 1);
by(zftac isValidSession_2' 1);
by(zetac (get_decl HSD.thy "SessionManager" 1) 1);
be conjE 1; be exE 1;
by(REPEAT (etac conjE 1));
by(case_tac "sid = Signature_generation_sid" 1);
   by(hyp_subst_tac 1);
```

```
   by(HINT "uid = x" (K all_tac) 1);
   by(rotate_tac ~3 1);
   by(Asm_full_simp_tac 1);
   by (zdtac (get_conj HSD.thy "SessionManager" 1) 1);
   br bexI 1;br X_in_SESSION_ID 2;
   br conjI 1; ba 2;
   by(rotate_tac ~1 1);
   by(Asm_full_simp_tac 1);

(* now the other case - essentially covered by lemma: *)
be AppendSignatureRecord_imp_nosid_nochange 1;
by(ALLGOALS Asm_simp_tac);
by(zetac (get_decl HSD.thy "SessionManager" 1) 1);
by(ALLGOALS stripS_tac);
be (stripS(get_conj HSD.thy "SessionManager" 1)) 2;
be (stripS(get_conj HSD.thy "SessionManager" 2)) 1;
by(HINT "SESSION_ID = UNIV" (fn _ => (rtac set_ext 6) THEN (Asm_simp_tac 6)) 5);
auto();
qed"GenerateSignatureL_not_accept_read_key_implies_inv";




zgoal HSD.thy
"LogoutL =+=> (signature_log' =  signature_log &           \
\              session_IDs' = session_IDs &               \
\              access_control_list' = access_control_list & \
\              pri_key_list' = pri_key_list)";
by(stripS_tac 1);
by(forward_tac [stripS (get_decl HSD.thy "LogoutL" 2)] 1);
by(forward_tac [stripS (get_decl HSD.thy "LogoutL" 3)] 1);
by(dtac (stripS (get_conj HSD.thy "LogoutL" 4)) 1);
by (convert2hol_tac [] 1);
qed"LogoutL_inv_state_components";

zgoal HSD.thy
"LogoutL =+=>                                                \
\ (uid : (dom session_table) /\\ Logout_ID : (dom (session_table %^ uid))) \
\ =+=>  uid ~: (dom session_table')";
by(stripS_tac 1);be conjE 1;
by(full_expand_schema_tac LogoutL_def 1);
by(res_inst_tac [("t","session_table'")] subst 1);
br FreeSessionInformation_deletes_sid2 2;
auto();
by(forw_inst_tac [("f","snd")] arg_cong 1);by(rotate_tac ~1 1);
by(forw_inst_tac [("f","snd")] arg_cong 2);by(rotate_tac ~1 2);
by(asm_full_simp_tac (HOL_ss addsimps [DELTA_def]) 3);
be conjE 3;
by(dtac (stripS (get_decl HSD.thy "SessionManager" 1)) 3);
by(ALLGOALS(asm_full_simp_tac (prod_ss)));
qed"LogoutL_delete_uid";


zgoal HSD.thy
"LogoutL =+=>                                                \
\ (uid : (dom session_table) /\\ Logout_ID ~: (dom (session_table %^ uid))) \
\ =+=>  uid : (dom session_table')";
by(stripS_tac 1);be conjE 1;
by(full_expand_schema_tac LogoutL_def 1);
by(res_inst_tac [("t","session_table'")] subst 1);
```

```
br FreeSessionInformation_deletes_sid3 2;
auto();
by(forw_inst_tac [("f","snd")] arg_cong 1);by(rotate_tac ~1 1);
by(forw_inst_tac [("f","snd")] arg_cong 2);by(rotate_tac ~1 2);
by(asm_full_simp_tac (HOL_ss addsimps [DELTA_def]) 3);
be conjE 3;
by(dtac (stripS (get_decl HSD.thy "SessionManager" 1)) 3);
by(ALLGOALS(asm_full_simp_tac (prod_ss)));
qed"LogoutL_keep_uid";


zgoal HSD.thy
"LogoutL =+=>  (uid : (dom session_table'))   \
\ =+=>  (session_table' %^ uid = session_table %^ uid)";
by(stripS_tac 1);
by(forward_tac [stripS (get_conj HSD.thy "LogoutL" 2)] 1);
by(dres_inst_tac [("f","snd")]arg_cong 1);
by(eres_inst_tac [("P","uid : ?X")] rev_mp 1);
by(ALLGOALS(Asm_full_simp_tac));
by(zstac (FreeSessionInformation_axdef RS DECL_D2) 1);
  by(forward_tac [stripS (get_decl HSD.thy "LogoutL" 1)] 1);
  by(asm_full_simp_tac (HOL_ss addsimps [DELTA_def]) 1);
  be conjE 1;
  by(zetac (get_decl HSD.thy "SessionManager" 1) 1);
by(case_tac "Logout_ID : dom (gen_un (ran session_table))" 1);
by(ALLGOALS(Asm_simp_tac));
by(asm_simp_tac (simpset() addsimps [Let_def,maplet_def,asSet_def,image_def]) 1);
by(asm_simp_tac (simpset() addsimps [Domain_def,rel_apply_def]) 1);
qed"LogoutL_session_table_inv";



(* Nop is really a Nop and does not change any component
   if the state. *)
zgoal HSD.thy
"NopOperationL =+=> (session_table' = session_table &       \
\              session_IDs' = session_IDs &                 \
\              signature_log' =  signature_log &             \
\              access_control_list' = access_control_list & \
\              pri_key_list' = pri_key_list)";
by(stripS_tac 1);
by(forward_tac [stripS (get_decl HSD.thy "NopOperationL" 1)] 1);
by(forward_tac [stripS (get_decl HSD.thy "NopOperationL" 2)] 1);
by(forward_tac [stripS (get_decl HSD.thy "NopOperationL" 3)] 1);
by (convert2hol_tac [] 1);
qed"NopOperationL_inv_state_components";


(* ********************************************************************** *)
(*      Project          : HSD security analysis
        Author           : B. Wolff
        Affiliation      : ETH Z\"urich
        This theory      : Wiring the Architecture of HSD
                           (Version based on Latex-ZETA format)
        $Date: 2002/12/11 14:35:46 $
        $Revision: 1.3 $
        Release          : 2.5                                          *)
(* ********************************************************************** *)

(* cd "holz";
   use_holz "HSDArch";
 *)
```

```
toToplevel HSDArch.axdefs;
toToplevel HSDArch.schemes;

Add_axdefs_TC (map snd HSDArch.axdefs);



(* Architectural decomposition theorem - If the combined system makes a step,
   then it must have one of the following 4 forms:
*)

zgoal HSDArch.thy
"System =+=>  ((%E DARMA @   AuthenticateUserW & AuthenticateUserL)   |  \
\              (%E DARMA @   GenerateSignatureW & NopOperationL)      |  \
\              (%E DARMA @   GenerateSignatureW & GenerateSignatureL) |  \
\              (%E DARMA @   LogoutW & LogoutL))";
by(stripS_tac 1);
by(full_expand_schema_tac System_def 1);
by(elim_sch_ex_tac 1);
be conjE 1;
by(full_expand_schema_tac ClientOperation_def 1);

(* Case-Distinction over Client Operations ... *)
be disjE 1;
be disjE 2;

(* Throw away superfluous disjoints in order to
   make proof state smaller ... *)
br disjI1 1;
br disjI2 2;
br disjI2 3;
br disjI2 3;
br disjI2 3;

(* Now we extract the commands the user sends ... *)
by(forward_tac [stripS (get_conj HSD.thy "AuthenticateUserW" 3)] 1);
by(forward_tac [stripS (get_conj HSD.thy "GenerateSignatureW" 3)] 2);
by(forward_tac [stripS (get_conj HSD.thy "LogoutW" 1)] 3);
(* for the case GenerateSignatureW, we make a case distinction over
   hash-Failure. *)
by(case_tac "message_63 : hashFailure_" 2);
by(rotate_tac ~1 2);
by(rotate_tac ~1 3);
by(ALLGOALS Asm_full_simp_tac);

(* Throw away superfluous disjoints in order to
   make proof state smaller ... *)
br disjI1 2;
br disjI2 3;
br disjI1 3;

(* Now, unfold the server and produce all cases ... *)
by(ALLGOALS (full_expand_schema_tac ServerOperation_def));
by(Safe_tac);
(* ... produces 16 cases. We extract the commands the
   Server may go and lead 12 cases to contradictions
   with what the Client wanted. *)
by(forward_tac [stripS (get_conj HSD.thy "GenerateSignatureL" 1)] 2);
by(forward_tac [stripS (get_conj HSD.thy "LogoutL" 1)] 3);
by(forward_tac [stripS (get_conj HSD.thy "NopOperationL" 1)] 4);

by(forward_tac [stripS (get_conj HSD.thy "AuthenticateUserL" 1)] 5);
```

```
by(forward_tac [stripS (get_conj HSD.thy "GenerateSignatureL" 1)] 6);
by(forward_tac [stripS (get_conj HSD.thy "LogoutL" 1)] 7);

by(forward_tac [stripS (get_conj HSD.thy "AuthenticateUserL" 1)] 9);
by(ALLGOALS Asm_full_simp_tac); (* Just exploit contradictions *)

by(forward_tac [stripS (get_conj HSD.thy "LogoutL" 1)] 4);
by(forward_tac [stripS (get_conj HSD.thy "NopOperationL" 1)] 5);
by(forward_tac [stripS (get_conj HSD.thy "AuthenticateUserL" 1)] 6);
by(forward_tac [stripS (get_conj HSD.thy "GenerateSignatureL" 1)] 7);
by(forward_tac [stripS (get_conj HSD.thy "NopOperationL" 1)] 9);
by(ALLGOALS Asm_full_simp_tac); (* Just exploit contradictions *)

(*
by(forward_tac [stripS (get_conj HSD.thy "NopOperationL" 1)] 4);
by(forward_tac [stripS (get_conj HSD.thy "NopOperationL" 1)] 6);
by(ALLGOALS Asm_full_simp_tac); (* Just exploit contradictions *)

does not work due to bug in get_conj (resp. expand_schema_tac.
which does simplification with prod_ss which already includes
set simplification - i.e. too much.

Workaround: use the results of the internal simplification ...

*)
by(forward_tac [stripS (get_conj HSD.thy "NopOperationL" 2)] 4);
by(forward_tac [stripS (get_conj HSD.thy "NopOperationL" 3)] 6);
by(ALLGOALS Asm_full_simp_tac); (* Just exploit contradictions *)

by(intro_sch_ex_tac 1);
br conjI 1; ba 1; ba 1;
by(ALLGOALS (fn x => TRY(rtac refl x)));
bd(stripS (get_decl HSD.thy "AuthenticateUserW" 4)) 1;
by (convert2hol_tac [] 1);

by(intro_sch_ex_tac 1);
br conjI 1; ba 1; ba 1;
by(ALLGOALS (fn x => TRY(rtac refl x)));
bd(stripS (get_decl HSD.thy "GenerateSignatureW" 4)) 1;
by (convert2hol_tac [] 1);

by(intro_sch_ex_tac 1);
br conjI 1; ba 1; ba 1;
by(ALLGOALS (fn x => TRY(rtac refl x)));
bd(stripS (get_decl HSD.thy "GenerateSignatureW" 4)) 1;
by (convert2hol_tac [] 1);

by(intro_sch_ex_tac 1);
br conjI 1; ba 1; ba 1;
by(ALLGOALS (fn x => TRY(rtac refl x)));
bd(stripS (get_decl HSD.thy "LogoutW" 3)) 1;
by (convert2hol_tac [] 1);
qed"SysArch_decomposition_theorem";

(* Although the proof technique is simple and straight-forward, the
   proof-states become quite large and very unpleasant to work with.
   Therefore, architectural re-wiring of this kind is better hidden
   inside this theorem ... *)



zgoal HSDArch.thy
```

```
"((%E DARMA @   AuthenticateUserW & AuthenticateUserL) |    \
\ (%E DARMA @   GenerateSignatureW & NopOperationL) |       \
\ (%E DARMA @   GenerateSignatureW & GenerateSignatureL) |  \
\ (%E DARMA @   LogoutW & LogoutL))  =+=>                    \
\ System";
by(stripS_tac 1);
by(full_expand_schema_tac System_def 1);
by(Safe_tac);

by(elim_sch_ex_tac 1);
be conjE 1;
by(intro_sch_ex_tac 1);
by(ALLGOALS (fn x => TRY(rtac refl x)));
bd(stripS (get_decl HSD.thy "AuthenticateUserW" 4)) 2;
by (convert2hol_tac [] 2);
by(full_expand_schema_tac ClientOperation_def 1);
by(full_expand_schema_tac ServerOperation_def 1);

by(elim_sch_ex_tac 1);
be conjE 1;
by(intro_sch_ex_tac 1);
by(ALLGOALS (fn x => TRY(rtac refl x)));
bd(stripS (get_decl HSD.thy "GenerateSignatureW" 4)) 2;
by (convert2hol_tac [] 2);
by(full_expand_schema_tac ClientOperation_def 1);
by(full_expand_schema_tac ServerOperation_def 1);

by(elim_sch_ex_tac 1);
be conjE 1;
by(intro_sch_ex_tac 1);
by(ALLGOALS (fn x => TRY(rtac refl x)));
bd(stripS (get_decl HSD.thy "GenerateSignatureW" 4)) 2;
by (convert2hol_tac [] 2);
by(full_expand_schema_tac ClientOperation_def 1);
by(full_expand_schema_tac ServerOperation_def 1);

by(elim_sch_ex_tac 1);
be conjE 1;
by(intro_sch_ex_tac 1);
by(ALLGOALS (fn x => TRY(rtac refl x)));
bd(stripS (get_decl HSD.thy "LogoutW" 3)) 2;
by (convert2hol_tac [] 2);
by(full_expand_schema_tac ClientOperation_def 1);
by(full_expand_schema_tac ServerOperation_def 1);
qed"SysArch_introduction_theorem";


(* ********************************************************************** *)
(*       Project          : HSD security analysis
         Author           : B. Wolff
         Affiliation      : ETH Z\"urich
         This theory      : Embedding Architecture into Kripke
                            Structure and Formalizing Security Requirements
                            over it
                            (Version based on Latex-ZETA format)
         $Date: 2002/12/11 14:35:46 $
         $Revision: 1.3 $
         Release          : 2.5                                          *)
(* ********************************************************************** *)

(* cd "holz";
   use_holz "Analysis";
 *)
```

```
toToplevel Analysis.axdefs;
toToplevel Analysis.schemes;

Add_axdefs_TC (map snd Analysis.axdefs);

Delsimps[No_Dom_Restr]; (* seems to be unmovable ... *)



val prems = goalw Analysis.thy [Traces_def,Init_def]
"!!t. t : Traces ==>                                                \
\     EX acces_control_list pri_key_list.                           \
\            acces_control_list : ACCESS_CONTROL_LIST &             \
\            pri_key_list : PRI_KEY_LIST &                          \
\            t %^ #0 = (acces_control_list, pri_key_list, {}, {}, {})";
by (convert2hol_tac [InitState_def,SessionManagerInit_def,AccessController_def,
                     HysteresisSignatureInit_def,AccessControllerInit_def] 1);
auto();
qed "traces_init_D1";
Addsimps[traces_init_D1];



zgoalw Analysis.thy [Traces_def]
"t : Traces =+=>   t : (%N ---> GlobalState)";
by(stripS_tac 1);
by (convert2hol_tac [] 1);
auto();
qed "traces_init_D2";
Addsimps[stripS traces_init_D2];


val prems = goalw Analysis.thy []
"!!t.[| t : Traces; i : %N |] ==>                                   \
\     EX acl_l pky_l sIDs s_tab sig_log. ((t %^ i) =                \
\               (acl_l,pky_l,sIDs,s_tab,sig_log) &                  \
\                SessionManager (sIDs,s_tab) &                      \
\                (HysteresisSignature sig_log &                     \
\                 AccessController (acl_l,pky_l))) ";
bd (stripS traces_init_D2) 1;
bd tfun_apply 1; ba 1;
by (convert2hol_tac [GlobalState_def] 1);
by(res_inst_tac [("p","t %^ i")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","y")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","ya")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yb")] PairE 1);
by(rotate_tac ~1 1);
auto();
qed "trace_GlobalState";


val prems = goalw Analysis.thy []
"!!t.[| t : Traces; i : %N |] ==>                                   \
\     EX acl_l pky_l sIDs s_tab sig_log. ((t %^ (i + #1)) =         \
\               (acl_l,pky_l,sIDs,s_tab,sig_log) &                  \
\                SessionManager (sIDs,s_tab) &                      \
\                (HysteresisSignature sig_log &                     \
\                 AccessController (acl_l,pky_l))) ";
```

```
br trace_GlobalState 1;
auto();
qed "trace_GlobalStateSuc";

(* Destruction rule that performs projection. *)
val prems = goalw Analysis.thy [Traces_def,Next_def]
"!!x. [| System x |] ==>                                                \
\       EX  MsgHash_I SID_O SID_I                                       \
\           acl acl' mess_I pwd_I pkl                                   \
\           pkl' result_O sIDs sIDs'                                    \
\           ses_id_O ses_id_I s_tab s_tab'                              \
\           sig_O sig_log sig_log' uid_I uname_I.                       \
\      x =  (MsgHash_I, SID_O, SID_I,                                   \
\           acl, acl', mess_I, pwd_I, pkl,                              \
\           pkl', result_O, sIDs, sIDs',                                \
\           ses_id_O, ses_id_I, s_tab, s_tab',                          \
\           sig_O, sig_log, sig_log', uid_I, uname_I)";
by(res_inst_tac [("p","x")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","y")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","ya")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yb")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yc")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yd")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","ye")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yf")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yg")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yh")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yi")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yj")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yk")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yl")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","ym")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yn")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yo")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yp")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yq")] PairE 1);
by(rotate_tac ~1 1);
by(res_inst_tac [("p","yr")] PairE 1);
by(rotate_tac ~1 1);
auto();
qed "System_Project";
```

```
(* The following big chunk extends the System Decomposition Theorem
   to a theorem over subsequent trace points: Provided that a property
   holds for four system transition scenarios, it holds for all of them
   (the other are impossible). *)


val [p1,p2,p3,p4,p5,p6] = goalw Analysis.thy [Traces_def,Next_def]
"[|t : Traces; i : %N;                                                \
\  !! acl acl' mess_I pwd_I pkl pkl' sIDs sIDs' s_tab s_tab'          \
\     sig_log sig_log'  SID_O  ses_id_O sig_O uid_I uname_I           \
\     x xa xb xc xd xe xf xg y.                                       \
\     [| t %^ i = (acl, pkl, sIDs, s_tab, sig_log);                   \
\        t %^ (i + #1) = (acl',pkl',sIDs',s_tab',sig_log');           \
\        SNAME DARMA (x,xa,xb,xc,xd,xe,xf,xg,y);                      \
\        SNAME AuthenticateUserW                                      \
\          (x,xa,xb,xc,xd,xe,xf,xg,y,pwd_I,ses_id_O,uid_I);           \
\        SNAME AuthenticateUserL                                      \
\         (x,xa,xb,xc,SID_O,xd,xe,xf,xg,y,acl,acl',pwd_I,             \
\          pkl,pkl',sIDs,sIDs',s_tab,s_tab',sig_log,sig_log',         \
\          uname_I)|]                                                 \
\     ==> P (acl, pkl, sIDs, s_tab, sig_log)                          \
\           (acl', pkl', sIDs', s_tab', sig_log');                    \
\                                                                     \
\  !! acl acl' mess_I pwd_I pkl pkl' sIDs sIDs' s_tab s_tab'          \
\     sig_log sig_log' ses_id_I sig_O                                 \
\     x xa xb xc xd xe xf xg y.                                       \
\     [| t %^ i = (acl, pkl, sIDs, s_tab, sig_log);                   \
\        t %^ (i + #1) = (acl',pkl',sIDs',s_tab',sig_log');           \
\        SNAME DARMA (x,xa,xb,xc,xd,xe,xf,xg,y);                      \
\        SNAME GenerateSignatureW                                     \
\           (x,xa,xb,xc,xd,xe,xf,xg,y,mess_I,ses_id_I,sig_O);         \
\        SNAME NopOperationL                                          \
\           (x,xa,xb,xc,xd,xe,xf,xg,y,acl,acl',pkl,pkl',sIDs,         \
\            sIDs', s_tab, s_tab', sig_log, sig_log') |]              \
\     ==> P (acl, pkl, sIDs, s_tab, sig_log)                          \
\           (acl', pkl', sIDs', s_tab', sig_log');                    \
\                                                                     \
\  !! acl acl' mess_I pwd_I pkl pkl' sIDs sIDs' s_tab s_tab'          \
\     sig_log sig_log' MsgHash_I SID_I  ses_id_I sig_O                \
\     x xa xb xc xd xe xf xg y.                                       \
\     [| t %^ i = (acl, pkl, sIDs, s_tab, sig_log);                   \
\        t %^ (i + #1) = (acl',pkl',sIDs',s_tab',sig_log');           \
\        SNAME DARMA (x,xa,xb,xc,xd,xe,xf,xg,y);                      \
\        SNAME GenerateSignatureW                                     \
\           (x,xa,xb,xc,xd,xe,xf,xg,y,mess_I,ses_id_I,sig_O);         \
\        SNAME GenerateSignatureL                                     \
\           (x,xa,xb,MsgHash_I,xc,SID_I,xd,xe,xf,xg,y,acl,            \
\            acl',pkl,pkl',sIDs,sIDs',s_tab,s_tab',sig_O,             \
\            sig_log, sig_log')                                       \
\     |]                                                              \
\     ==> P (acl, pkl, sIDs, s_tab, sig_log)                          \
\           (acl', pkl', sIDs', s_tab', sig_log');                    \
\                                                                     \
\  !! acl acl' mess_I pwd_I pkl pkl' sIDs sIDs' s_tab s_tab'          \
\     sig_log sig_log' SID_I  result_O ses_id_I                       \
\     x xa xb xc xd xe xf xg y.                                       \
\     [| t %^ i = (acl, pkl, sIDs, s_tab, sig_log);                   \
\        t %^ (i + #1) = (acl',pkl',sIDs',s_tab',sig_log');           \
\        SNAME DARMA (x,xa,xb,xc,xd,xe,xf,xg,y);                      \
\        SNAME LogoutW (x,xa,xb,xc,xd,xe,xf,xg,y,result_O,ses_id_I);  \
\        SNAME LogoutL                                                \
\             (x,xa,xb,xc,SID_I,xd,xe,xf,xg,y,acl,acl',pkl,           \
```

```
\              pkl',result_0,sIDs,sIDs',s_tab,s_tab',sig_log,    \
\              sig_log')    |]                                    \
\     ==> P (acl, pkl, sIDs, s_tab, sig_log)                     \
\           (acl', pkl', sIDs', s_tab', sig_log')                \
\                                                                \
\|] ==>                                                          \
\                                                                \
\    P (t %^ i) (t %^ (i + #1))";
by(cut_facts_tac [p1,p2] 1);
by (convert2hol_tac [SSet_def] 1);
auto();
by(eres_inst_tac [("x","i")]  ballE 1);
auto();
by (forward_tac [System_Project] 1);
auto();
by(dtac(asm_full_simplify Z2HOL_ss(stripS SysArch_decomposition_theorem))1);
auto();
by(rtac(asm_full_simplify Z2HOL_ss p3) 1);
auto();
by(rtac(asm_full_simplify Z2HOL_ss p4) 1);
auto();
by(rtac(asm_full_simplify Z2HOL_ss p5) 1);
auto();
by(rtac(asm_full_simplify Z2HOL_ss p6) 1);
auto();
qed"State_Transition_Cases";



(* ************************************************************************ *)
(*                                                                          *)
(* Global Invariants ...                                                    *)
(*                                                                          *)
(* ************************************************************************ *)


val prems = goalw Analysis.thy []
"!!t. [|t:Traces ; i : %N |] ==>  \
\ fst(t %^ #0) = fst(t %^ i) &\
\ fst(snd(t %^ #0)) = fst(snd(t %^ i))";
by(res_inst_tac [("x","i")] naturals_induct 1);
ba 1;
by(ALLGOALS(Asm_simp_tac));
be State_Transition_Cases 1;
by(ALLGOALS(Asm_simp_tac));
by(zdtac AuthenticateUserL_inv_state_components 1);
by(zdtac NopOperationL_inv_state_components 1);
by(zdtac GenerateSignatureL_inv_state_components 1);
by(zdtac LogoutL_inv_state_components 1);
qed"acl_and_pkl_inv0";


(* Now we bring this in a form that can be nicely
   printed:

 1. !!t. [| t : Traces; i : %N |]
         ==> (t %^ #0).access_control_list = (t %^ i).access_control_list /\
             (t %^ #0).private_key_list = (t %^ i).private_key_list
 *)
val prems = goalw Analysis.thy []
"!!t. [|t:Traces ; i : %N |] ==>                                 \
\  PROJ(t %^ #0)(fst)(''access_control_list'') =                 \
```

```
\  PROJ(t %^ i) (fst)(''access_control_list'') &              \
\  PROJ(t %^ #0)(% x. fst(snd(x)))(''private_key_list'') =    \
\  PROJ(t %^ i) (% x. fst(snd(x)))(''private_key_list'')";
by (convert2hol_tac [SSet_def] 1);
br  acl_and_pkl_inv0 1;
auto();
qed"acl_and_pkl_inv";



(* printed nicely, this system invariant looks as follows:

    "[| ?t : Traces; ?i : %N |]
    ==> dom ((?t %^ ?i).signature_log) <=
        dom ((?t %^ (?i + #1)).signature_log)"

    This global system invariant motivates the precise definition for
    siglogChanges. Strictly speaking, the case

       uid : dom(s1.signature_log) & uid ~: dom(s2.signature_log)

    would represent a "change" of the log. Due to signature_log_mono,
    this case is inherently impossible.

 *)

val prems = goalw Analysis.thy []
"!!t. [|t:Traces ; i : %N |] ==>                                    \
\  dom(PROJ(t %^ i)       (% x. snd(snd(snd(snd(x)))))(''signature_log'')) <= \
\  dom(PROJ(t %^ (i + #1))(% x. snd(snd(snd(snd(x)))))(''signature_log''))";
by (convert2hol_tac [SSet_def] 1);
be State_Transition_Cases 1;
by(ALLGOALS(Asm_simp_tac));
by(zdtac AuthenticateUserL_inv_state_components 1);
by(zdtac NopOperationL_inv_state_components 1);
by(zftac GenerateSignatureL_inv_state_components 1);
by(zdtac GenerateSignatureL_siglog_mono 1);
by(zdtac LogoutL_inv_state_components 1);
qed"signature_log_mono";


val prems = goalw Analysis.thy []
"!!t. [|t:Traces ; i : %N |] ==>                                    \
\  dom(PROJ(t %^ i)       (% x. snd(snd(snd(snd(x)))))(''signature_log'')) <= \
\  dom(PROJ(t %^ (i + $# z))(% x. snd(snd(snd(snd(x)))))(''signature_log''))";
by(induct_tac "z" 1);
by(ALLGOALS(Asm_simp_tac));
be subset_trans 1;
by(simp_tac (HOL_ss addsimps [zsuc_def]) 1);
by(res_inst_tac [("t"," $# 1")] subst 1);
be signature_log_mono 2;
by(ALLGOALS(Asm_simp_tac));
qed"signature_log_mono_trace";


(* nicely printed, the following theorem looks as:

!!t. [| t : Traces; i : %N |]
     ==> (t %^ i).session_IDs <= (t %^ (i + #1)).session_IDs

  i.e. the system is monotone in the session_IDs parameter. *)
```

```
val prems = goalw Analysis.thy []
"!!t. [|t:Traces ; i : %N |] ==>                                          \
\  PROJ(t %^ i)       (% x. fst(snd(snd(x))))(''session_IDs'') <=     \
\  PROJ(t %^ (i + #1))(% x. fst(snd(snd(x))))(''session_IDs'')";
by (convert2hol_tac [SSet_def] 1);
be State_Transition_Cases 1;
by(ALLGOALS(Asm_simp_tac));
by(zdtac (get_conj HSD.thy "AuthenticateUserL" 4) 1);
by(zdtac  NopOperationL_inv_state_components 2);
by(zdtac  GenerateSignatureL_inv_state_components 2);
by(zdtac  LogoutL_inv_state_components 2);
auto();
by(asm_simp_tac (simpset() addsimps [Let_def] addsplits [expand_if]) 1);
qed"session_IDs_mono";



(* More general than no_siglogChanges_at_init, in fact a
   semantic characterization of this event predicate :

   !!t n uid.
      [| t : Traces; n : %N; uid ~: dom ((t %^ n).session_table) |]
      ==> (t %^ n, t %^ (n + #1)) ~: siglogChanges %^ uid

  *)
val [] = goalw Analysis.thy []
"!!t n uid.                                                              \
\   [| t : Traces; n : %N;                                               \
\      uid ~: dom (PROJ (t %^ n)                                         \
\                  (% x. fst(snd(snd(snd(x)))))                          \
\                  ''session_table'')                                    \
\   |] ==> (t %^ n, t %^ (n + #1)) ~: siglogChanges %^ uid";
by(zstac (siglogChanges_axdef RS DECL_D2) 1);
by(zftac traces_init_D2 1);
by(zftac traces_init_D2 2);
by(ALLGOALS(asm_simp_tac (simpset() addsimps[tfun_apply])));
br State_Transition_Cases 1;
ba 1;
by(Asm_full_simp_tac 1);
by(zdtac AuthenticateUserL_inv_state_components 1);
by(zdtac NopOperationL_inv_state_components 2);
by(zftac GenerateSignatureL_inv_state_components 2);
by(zdtac LogoutL_inv_state_components 3);
by (ALLGOALS(convert2hol_tac []));
auto(); (* exploit architectural contradictions ... *)
bd (asm_full_simplify Z2HOL_ss
          (stripS GenerateSignatureL_implies_not_siglogChanges)) 1;
bd (asm_full_simplify Z2HOL_ss
          (stripS GenerateSignatureL_implies_not_siglogChanges)) 3;
auto();
qed"uid_not_in_session_table_implies_no_siglogChanges";


(* Corollary: there will never be a siglogchange at the beginning
   of a trace *)
val prems = goalw Analysis.thy []
"[|t:Traces |] ==> (t %^ #0, t %^ #1) ~: siglogChanges %^ uid";
by(cut_facts_tac prems 1);
by(res_inst_tac [("t","#1")] subst 1);
br uid_not_in_session_table_implies_no_siglogChanges 2;
bd traces_init_D1 4;
auto();
```

```
by (convert2hol_tac [] 1);
qed"no_siglogChanges_at_init";



(* nicely printed:

   [| t:Traces;i:%N |] ==> dom ((t %^ i).signature_log) <= dom ((t %^ i).pri_key_list)
   [| t : Traces; i : %N |]
     ==> dom ((t %^ i).session_table) <= dom ((t %^ i).pri_key_list) /\
         dom ((t %^ i).signature_log) <= dom ((t %^ i).pri_key_list)
 *)

val prems = goalw Analysis.thy []
"!!t. [|t:Traces ; i : %N |] ==>                                          \
\  dom(PROJ(t %^ i)(% x. fst(snd(snd(x)))))(''session_table'')) <=  \
\  dom(PROJ(t %^ i)(% x. fst(snd(x)))(''pri_key_list'')) &              \
\  dom(PROJ(t %^ i)(% x. snd(snd(snd(snd(x)))))(''signature_log'')) <=  \
\  dom(PROJ(t %^ i)(% x. fst(snd(x)))(''pri_key_list''))";
by(res_inst_tac [("x","i")] naturals_induct 1);
ba 1;
bd traces_init_D1 2;
by (ALLGOALS (convert2hol_tac [SSet_def]));
by(eres_inst_tac [("P","?X & ?Y")] rev_mp 1);
by(HINT "$# 1 = #1" (Asm_simp_tac) 1);
by(asm_simp_tac HOL_ss 1);
be State_Transition_Cases 1;
by(ALLGOALS(Asm_simp_tac));
be exE 5; be conjE 5; be exE 5;
by(Asm_full_simp_tac 5);
by(zftac AuthenticateUserL_inv_state_components 1);
by(zdtac NopOperationL_inv_state_components 2);
by(zftac GenerateSignatureL_inv_state_components 2);
by(zftac LogoutL_inv_state_components 3);
br impI 2;
by(zdtac GenerateSignatureL_implies_pri_key_list_bounds 2);
by(ALLGOALS Asm_full_simp_tac);

(* Consider LogoutL: use definition of session_table',
   unfold FreeSessionInformation, and prove that
   removing entries from s_tab does not conflict with the fact
   that s_tab is bound by pkl. *)
by(zftac (get_decl HSD.thy "LogoutL" 1) 2);
by(asm_full_simp_tac (HOL_ss addsimps[DELTA_def]) 2);
by(REPEAT (etac conjE 2));
by(zdtac (get_decl HSD.thy "SessionManager" 1) 2);
by(zdtac (get_conj HSD.thy "LogoutL" 2) 2);
by(dres_inst_tac [("f","snd"),("x","(xc,s_tab')")] arg_cong 2);
by(ALLGOALS Asm_full_simp_tac);
by(zstac (FreeSessionInformation_axdef RS DECL_D2) 2);
by(simp_tac (simpset() addsimps  [Let_def,maplet_def,asSet_def,image_def]
                       addsplits [expand_if]) 2);
by(thin_tac "s_tab' = ?X" 2);
by(Blast_tac 2);

(* Consider AuthenticateUser: an entry is made, but this results
   from fst(CheckValidOfSession) and thus is bound by  dom pkl. *)
(* Saturation >>> *)
by(zftac (get_decl HSD.thy "AuthenticateUserL" 1) 1);
by(zftac (get_decl HSD.thy "AuthenticateUserL" 2) 1);
by(zftac (get_decl HSD.thy "AuthenticateUserL" 3) 1);
by(asm_full_simp_tac (HOL_ss addsimps[XI_def,DELTA_def]) 1);
```

```
by(REPEAT (etac conjE 1));
by(zftac (get_decl HSD.thy "SessionManager" 2) 1);
by(zdtac (get_decl HSD.thy "SessionManager" 1) 1);
by(zdtac (get_decl HSD.thy "DARMA" 4) 1);
(* <<< Saturation  *)
(* now comes the sun *)
by(zftac (get_conj HSD.thy "AuthenticateUserL" 3) 1);
by(zdtac (get_conj HSD.thy "AuthenticateUserL" 2) 1);
by(hyp_subst_tac 1);
by(thin_tac "s_tab' = ?X" 1);
by(thin_tac "t %^ x = ?X" 1);
by(thin_tac "t %^ (x+ #1) = ?X" 1);
by(thin_tac "xa = ?X" 1);
by(forward_tac [hash_axdef RS DECL_D1 RS conjunct1 RS tfun_apply] 1);
by(zftac (get_decl HSD.thy "AccessController" 1) 1);
by(zftac (get_decl HSD.thy "AccessController" 2) 1);
by(zdtac (get_conj HSD.thy "AccessController" 1) 1);
by(zstac (AuthenticateUser_axdef RS DECL_D2) 1);
by(asm_simp_tac (simpset() addsimps [AUTH_ERRORS_def,maplet_def]
                           addsplits [expand_if]) 1);
qed"dom_tabs_contained_in_dom_pri_key_list";




(* now comes the first half of the HSD_1 proofs:
   if a user never logs in, he will not be in the session table.

   "[| t : Traces; n : %N |]
    ==> (! k: #0 .. n - #1. (t%^k,t%^(k + #1)) ~: userDoesLogin%^uid) =+=>
        uid ~: dom ((t%^n).session_table)"

   This corresponds to what has been proven in the SPIN case study.

 *)
(* XXX *)
val prems = goalw Analysis.thy []
"!!t. [|t:Traces ; n : %N |] ==>                                                \
\     (!k : #0 .. n - #1.  (t %^ k, t %^ (k + #1)) ~: userDoesLogin %^ uid) -->  \
\     uid ~: dom (PROJ (t %^ n)                                                 \
\                      (% x. fst(snd(snd(snd(x)))))
\
\                      ''session_table'')";
by(res_inst_tac [("x","n")] naturals_induct 1); ba 1;
by(ALLGOALS Asm_full_simp_tac);
(* First part: *Base* exploit inital state. *)
bd traces_init_D1 2;
by (convert2hol_tac [] 2);
auto();

(* Second part: (* Step *). A: exploit contradicting assumptions: *)
by(eres_inst_tac [("x","k")] ballE 1);
auto();
by(eres_inst_tac [("Q","k ~: ( #0 .. ?X)")] contrapos2 1);
by(ALLGOALS Asm_simp_tac);
br numb_range_mem_subset2 1; ba 2;
by(ALLGOALS Asm_simp_tac);

(* Second part: (* Step *). B: specialize hypothesis to case k=x. *)
by(eres_inst_tac [("x","x")] ballE 1);
by(eres_inst_tac [("Q","x ~: ( #0 .. ?X)")] contrapos2 2);
```

```
by(ALLGOALS Asm_simp_tac);
by(asm_full_simp_tac (HOL_ss addsimps [numb_range_def,in_naturals RS sym]) 2);
by(Asm_simp_tac 2);
br zequalD1 2;
by (stac zadd_assoc 2);
by(Asm_simp_tac 2);

(* Second part: (* Step *). B: unfold userDoesLogin and bring
   it to self-contradiction with the hypothesis. *)
bd pair_rel_dom 1;
by(rotate_tac ~1 1);
be contrapos2 1;
by(Asm_full_simp_tac 1);
by(rotate_tac ~1 1);
be rev_mp 1;
by(rotate_tac ~1 1);
be rev_mp 1;
by(zstac (userDoesLogin_axdef RS DECL_D2) 1);
by(zdtac traces_init_D2 1);
by(zdtac traces_init_D2 2);
by(ALLGOALS(asm_simp_tac (simpset() addsimps[tfun_apply])));
qed"HSD_1a_core";


val [] = goalw Analysis.thy [HSD_1a_def,SSet_def,asSet_def,image_def]
"y : HSD_1a";
auto();by(tc_tac 1);
by(rotate_tac ~1 1);
be contrapos2 1;
br uid_not_in_session_table_implies_no_siglogChanges 1;
by(ALLGOALS(Asm_full_simp_tac));
br (HSD_1a_core RS mp) 1;
by(ALLGOALS(Asm_full_simp_tac));
qed"HSD_1a";

(* 18.9. 2004, 10:47 *)


val prems = goalw Analysis.thy []
"!!t. [|t:Traces ; n : %N |] ==>                                              \
\     (! k: #0 .. n - #1.                                                      \
\            (t %^ k, t %^ (k + #1)) ~: userDoesLogin %^ uid |                 \
\            (? j: k + #1 .. n - #1. (t%^ j, t%^ (j+ #1)):userDoesLogout %^ uid))\
\      -->      uid ~: dom (PROJ (t %^ n)                                     \
\                                (% x. fst(snd(snd(snd(x)))))                 \
\                                ''session_table'')";
by(res_inst_tac [("x","n")] naturals_induct 1); ba 1;
by(ALLGOALS Asm_full_simp_tac);
(* First part: *Base* exploit initial state. *)
bd traces_init_D1 2;
by (convert2hol_tac [] 2);
auto();

(* Second part: (* Step *). A: exploit contradicting assumptions: *)
by(eres_inst_tac [("x","k")] ballE 1);
by(ALLGOALS Asm_full_simp_tac);
be bexE 1;
by(case_tac "j=x" 1);
by(rotate_tac ~2 1);
be contrapos2 1;
by(zstac (userDoesLogout_axdef RS DECL_D2) 1);
by(zftac traces_init_D2 1);
```

```
by(zftac traces_init_D2 2);
by(ALLGOALS(asm_simp_tac (simpset() addsimps[tfun_apply])));
by(eres_inst_tac [("x","j")] ballE 1);
by(Asm_full_simp_tac 1);
by(HINT "j ~: ( k + #1 .. x + #1 + #~01)" (K all_tac) 1);
by(Asm_full_simp_tac 1);
by(asm_full_simp_tac (simpset() addsimps [numb_range_def]) 1);
auto();
by(rotate_tac 7 1);
be swap 1;
by(rotate_tac 3 1);
bd zle_imp_zless_or_eq 1;
be disjE 1;
by(res_inst_tac [("t","x + #~01")] subst 1);
by(res_inst_tac [("m1","x")]  (less_zpred_eq_le RS iffD2) 2);
by(ALLGOALS Asm_simp_tac);
by(res_inst_tac [("t","x")] subst 2);
ba 3;
by (stac zadd_assoc 2);
by(Asm_simp_tac 2);
by(res_inst_tac [("t","x + #~01")] subst 1);
by(stac zsuc_zpred 2); br refl 2;
by(asm_full_simp_tac (HOL_ss addsimps [zpred_def]) 1);
auto();
by(HINT "x + #1 + #~01 = x" (K all_tac) 1);
by(Asm_full_simp_tac 1);
by (stac zadd_assoc 1);
by(Asm_simp_tac 1);

be swap 1;
br numb_range_mem_subset2 1; ba 2;
by(ALLGOALS Asm_simp_tac);


(* Second part: (* Step *). B: specialize hypothesis to case k=x. *)
by(eres_inst_tac [("x","x")] ballE 1);
by(eres_inst_tac [("Q","x ~: ( #0 .. ?X)")] contrapos2 2);
by(ALLGOALS Asm_simp_tac);
by(asm_full_simp_tac (HOL_ss addsimps [numb_range_def,in_naturals RS sym]) 2);
by(Asm_simp_tac 2);
br zequalD1 2;
by (stac zadd_assoc 2);
by(Asm_simp_tac 2);


(* Second part: (* Step *). B: unfold userDoesLogin and bring
   it to self-contradiction with the hypothesis. *)
bd pair_rel_dom 1;
by(rotate_tac ~1 1);
be contrapos2 1;
by(Asm_full_simp_tac 1);
be disjE 1;

by(rotate_tac ~1 1);
be rev_mp 1;
by(rotate_tac ~1 1);
be rev_mp 1;
by(zstac (userDoesLogin_axdef RS DECL_D2) 1);
by(zftac traces_init_D2 1);
by(zftac traces_init_D2 2);
by(ALLGOALS(asm_simp_tac (simpset() addsimps[tfun_apply])));
```

```
(* self contradiction : no userDoesLogout possible *)
be bexE 1;
by(HINT "x + #1 + #~01 = x" (K all_tac) 1);
by(rotate_tac ~1 1);
by(Asm_full_simp_tac 1);
by (stac zadd_assoc 2);
by(Asm_simp_tac 2);
by(asm_full_simp_tac (HOL_ss addsimps [numb_range_def]) 1);
auto();
qed"HSD_1b_core";
(*Sun Sep 19 19:40:59 MEST 2004 *)



val [] = goalw Analysis.thy [HSD_1b_def,SSet_def,asSet_def,image_def]
" y : HSD_1b";
auto();by(tc_tac 1);
be contrapos2 1;back();back();
by(Asm_full_simp_tac 1);
br uid_not_in_session_table_implies_no_siglogChanges 1;
by(ALLGOALS(Asm_full_simp_tac));
br (HSD_1b_core RS mp) 1;
by(ALLGOALS(Asm_full_simp_tac));
qed"HSD_1b";



val [] = goalw Analysis.thy [HSD_2a_def,SSet_def,asSet_def,image_def] "y : HSD_2a";
by(Asm_simp_tac 1);by(tc_tac 1);
by(stripS_tac 1);
by(rotate_tac ~1 1);
be rev_mp 1;
by(zstac (siglogChangedTo_axdef RS DECL_D2) 1);
by(zstac (signatureIsGenerated_axdef RS DECL_D2) 3);
by(zftac traces_init_D2 1);
by(zftac traces_init_D2 2);
by(zftac traces_init_D2 3);
by(zftac traces_init_D2 4);
by(ALLGOALS(asm_simp_tac (simpset() addsimps[tfun_apply])));

br State_Transition_Cases 1;
by(ALLGOALS(asm_simp_tac Z2HOL_ss));
by(zdtac AuthenticateUserL_inv_state_components 1);
by(zdtac NopOperationL_inv_state_components 1);
by(zftac GenerateSignatureL_inv_state_components 1);
by(zdtac LogoutL_inv_state_components 2);
by(case_tac "uid : dom s_tab" 1);
by(zdtac GenerateSignatureL_implies_not_siglogChanges 2);
be disjE 2;
by(ALLGOALS(Asm_simp_tac));
br impI 1;
by(REPEAT (etac conjE 1));
bd dom_tabs_contained_in_dom_pri_key_list 1; ba 1;

br conjI 1; br conjI 2;
br ballI 1;
by(zetac GenerateSignatureL_and_siglogChanges_implies_inv_others 1);
by(hyp_subst_tac 4);
by(zrtac GenerateSignatureL_and_siglogChanges_implies_prikey_use 4);
by (ALLGOALS(convert2hol_tac []));
auto();
```

```
qed"HSD_2a";
(* Mon Oct 18 14:20:35 MEST 2004 *)


goalw Analysis.thy [] "? y z. (($# x) = ($# y + $# z))";
by(res_inst_tac [("x","(0)::nat")] exI 1);
by(res_inst_tac [("x","(x)::nat")] exI 1);
auto();
qed"HSD_3_aux1";


goalw Analysis.thy []
" !!t.                                                                          \
\ [| t : Traces; n : %N;  uid : dom ssn_tbl; sid : dom (ssn_tbl %^ uid);       \
\    uid : dom slog;                                                           \
\    slog = PROJ(t %^ n)(% x. snd(snd(snd(snd(x))))(''signature_log'');        \
\    ssn_tbl = PROJ (t %^ n) (% x. fst(snd(snd(snd(x)))) ''session_table'';    \
\    accept_read_prikey ~= PROJ (ssn_tbl %^ uid %^ sid) fst ''pkra'' |] \
\ ==> (t %^ n, t %^ (n + #1)) ~: siglogChangedTo %^ (uid, sig)";
by(zstac (siglogChangedTo_axdef RS DECL_D2) 1);
by(zftac traces_init_D2 1);
by(zftac traces_init_D2 2);
by(ALLGOALS(asm_simp_tac (simpset() addsimps[tfun_apply])));
by(rotate_tac 5 1);
be rev_mp 1;be rev_mp 1;
br State_Transition_Cases 1;
by(ALLGOALS(asm_simp_tac Z2HOL_ss));
by(ALLGOALS(strip_tac ));
by(ALLGOALS(hyp_subst_tac));
by(zdtac AuthenticateUserL_inv_state_components 1);
by(zdtac NopOperationL_inv_state_components 1);
by(zftac GenerateSignatureL_inv_state_components 1);
by(zdtac LogoutL_inv_state_components 2);
by(ALLGOALS(Asm_simp_tac));

(* inserting * the lemma * here that does the work at the
   data level : *)
by(zdtac GenerateSignatureL_not_accept_read_key_implies_inv 1);
br conjI 1; ba 1;
br conjI 1; ba 1;
by (convert2hol_tac [] 1);
auto();
qed"HSD_3_inv_implies_post";



goalw Analysis.thy []
"!!t.[| t : Traces; k : %N; uid : USER_ID;
\
\       sig : SIGNATURE; sig' : SIGNATURE;
\
\       ssn_tbl = PROJ (t %^(k + #1)) (% x. fst(snd(snd(snd(x))))''session_table'';
\
\       uid : dom ssn_tbl & sid : dom(ssn_tbl %^ uid) &
\
\       accept_read_prikey ~= PROJ (ssn_tbl %^ uid %^ sid) fst ''pkra''
\
\    |]
\
\ ==>  (! j: k + #1 .. k + $# n. (t %^ j, t %^ (j + #1)) ~: userDoesLogout %^ uid)
\
\      --> (let ssn_tbl'=PROJ(t %^(k + $#n + #1))(% x. fst(snd(snd(snd(x))))''session_table''\
```

```
\           in  (uid : dom ssn_tbl' & sid : dom(ssn_tbl' %^ uid) &
\
\                accept_read_prikey ~= PROJ (ssn_tbl' %^ uid %^ sid) fst ''pkra''))";
by(REPEAT (etac conjE 1));
by(induct_tac "n" 1);
(* base case : *)
  br impI 1;
  by(thin_tac "Ball  ?X ?P" 1);
  by(asm_full_simp_tac (simpset() addsimps
                        [zadd_left_commute,zadd_commute,zadd_assoc,Let_def]
                        delsimps [inj_zint]) 1);
(* step: *)
  (* synchronize preconditions *)
  by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                            [numb_range_def, Ball_def])));
  br impI 1;be impE 1; br allI 1;
  by(eres_inst_tac [("x","x")] allE 1);
  by(Asm_simp_tac 1);
  by(eres_inst_tac [("x","zsuc (k + $# n)")] allE 1);
  by(thin_tac "ssn_tbl = ?X" 1);
  by(thin_tac "uid : dom ?X" 1);
  by(thin_tac "sid : dom ?X" 1);
  by(thin_tac "accept_read_prikey ~= ?X" 1);
  by(asm_full_simp_tac (simpset() addsimps
                        [zadd_left_commute,zadd_commute,zadd_assoc,Let_def]
                        delsimps [inj_zint]) 1);
  by(asm_full_simp_tac (HOL_ss addsimps
                        [zsuc_def, zadd_left_commute,zadd_commute,zadd_assoc]) 1);
  by(Asm_full_simp_tac 1);
  be impE 1;
  br (zle_refl RS zadd_zle_mono) 1;
  by (stac zadd_commute 1);
  br Nat_zle_zadd 1; br zle_refl 2; br naturalsI 1; br refl 1;

  (* now load the premises into the decomposition thm. *)
  by(rotate_tac ~2 1);
  be rev_mp 1; be rev_mp 1;
  by(res_inst_tac [("s","#1 + #1"),("t","#2")] subst 1);
  by(Asm_simp_tac 1);
  by(asm_full_simp_tac (HOL_ss addsimps
                        [zadd_assoc RS sym]) 1);
  by(zstac (userDoesLogout_axdef RS DECL_D2) 1);
  by(zftac traces_init_D2 1);
  by(zftac traces_init_D2 2);
  by(ALLGOALS(asm_simp_tac (simpset() addsimps[tfun_apply])));
  (* now standard decomposition a la carte ... *)
  br State_Transition_Cases 1;
  by(ALLGOALS(asm_simp_tac Z2HOL_ss));
  by(ALLGOALS(strip_tac ));
    (* side condition :  k + $# n + #1 : %N *)
    br Nat_zadd 1; br Nat_zadd 1;
    by(ALLGOALS(Asm_simp_tac ));

  by(zftac AuthenticateUserL_inv_state_components 1);
  by(zdtac NopOperationL_inv_state_components 2);
  by(zftac GenerateSignatureL_inv_state_components 2);
  by(zftac LogoutL_inv_state_components 3);
  by(ALLGOALS(fn x => (REPEAT(etac conjE x))));
  by(ALLGOALS(hyp_subst_tac));
  by(ALLGOALS(Asm_full_simp_tac));

  by(zdtac LogoutL_session_table_inv 3);
```

```
  by(Asm_full_simp_tac 3);
  by(zdtac GenerateSignatureL_not_accept_read_key_implies_inv 2);
  by (convert2hol_tac [] 2);
  by(Blast_tac 2);
  by(Asm_full_simp_tac 2);
  by(zdtac AuthenticateUserL_uid_auth_implies_session_table_inv 1);
  auto();
qed"HSD_3_core";


goalw Analysis.thy []
" !!t.
\
\ [| t : Traces; n : %N;
\
\    slog = PROJ(t %^ (n + #1))(% x. snd(snd(snd(snd(x)))))(''signature_log'');
\
\    ssn_tbl = PROJ (t %^ (n + #1)) (% x. fst(snd(snd(snd(x))))) ''session_table'';
\
\    (t %^ n, t %^ (n + #1)) : siglogChangedTo %^ (uid, sig) |]
\
\ ==> ? sid. uid : dom slog & uid : dom ssn_tbl & sid : dom(ssn_tbl %^ uid) &
\
\            accept_read_prikey ~= PROJ (ssn_tbl %^ uid %^ sid) fst ''pkra'' ";
by(rotate_tac 2 1);
be rev_mp 1;be rev_mp 1;be rev_mp 1;
by(zstac (siglogChangedTo_axdef RS DECL_D2) 1);
by(zftac traces_init_D2 1);
by(zftac traces_init_D2 2);
by(ALLGOALS(asm_simp_tac (simpset() addsimps[tfun_apply])));
br State_Transition_Cases 1;
by(ALLGOALS(asm_simp_tac Z2HOL_ss));
by(ALLGOALS(strip_tac ));
by(ALLGOALS(hyp_subst_tac));
by(zdtac AuthenticateUserL_inv_state_components 1);
by(zdtac NopOperationL_inv_state_components 2);
by(zftac GenerateSignatureL_inv_state_components 3);
by(zdtac LogoutL_inv_state_components 4);
by(ALLGOALS(fn x => (REPEAT(etac conjE x))));
by(ALLGOALS(hyp_subst_tac));
by(ALLGOALS(Asm_full_simp_tac));

(* prove first part of the invariant:
   uid : dom slog. This is essentially a consequence of
   siglogChangedTo and  GenerateSignatureL_siglog_mono *)
br conjI 1;
by(zftac GenerateSignatureL_siglog_mono 1);
by(Blast_tac 1);
(* prove crucial second part of the invariant:
   uid : dom ssn_tbl and existence of unique sids with
   accept_read_prikey ~= (ssn_tbl %^ uid %^ sid).pkra. This is essentially a
   consequence of GenerateSignatureL_implies_no_accept_key that does
   the real work at the data modeling level. *)
by(zftac GenerateSignatureL_siglogChanges_charn2 1);
by(Blast_tac 1);
br conjI 1;
by(Blast_tac 1);
br exI 1;
br conjI 1;
by(Blast_tac 1);
by(zdtac GenerateSignatureL_implies_no_accept_key 1);
by (convert2hol_tac [] 2);
```

```
auto();
qed "post_implies_HSD_3_inv";



val [] = goalw Analysis.thy [HSD_3_def,SSet_def,asSet_def,image_def] "y : HSD_3";
by(Asm_simp_tac 1);
by(stripS_tac 1);
(* index rectification *)
by(HINT "#0 <= k & k <= n + #~01" (K all_tac) 1);
by(asm_full_simp_tac (simpset() addsimps [numb_range_def,in_naturals,zless_eq_zadd]) 2);
by(REPEAT (etac conjE 1));
by(thin_tac "k : ?X" 1);
by(rotate_tac ~1 1);
bd(zless_eq_zadd RS iffD1) 1;
by(REPEAT (etac exE 1));
by(dres_inst_tac [("f","% x . (x::int) + #1")] arg_cong 1);
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                          [zadd_left_commute,zadd_commute,zadd_assoc]
                          delsimps [inj_zint])));
by(hyp_subst_tac 1);
by(zftac post_implies_HSD_3_inv 1);
  by(asm_full_simp_tac (simpset() addsimps [in_naturals]) 1);
by(REPEAT (etac exE 1));
by(REPEAT (etac conjE 1));
by(thin_tac "(t %^ k, ?X) : ?Y" 1);
by(eres_inst_tac [("Q","(?xa,?ya) : ?Y")] contrapos2 1);
by(ALLGOALS(Asm_full_simp_tac));
by(zftac HSD_3_core 1);
  by(asm_full_simp_tac (simpset() addsimps [in_naturals]) 1);
  by(Blast_tac 1);

by(asm_full_simp_tac (simpset() addsimps [Let_def]) 1);
by(REPEAT (etac conjE 1));
br HSD_3_inv_implies_post 1;
br refl 7; br refl 6;

by(ALLGOALS(Asm_full_simp_tac));
by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                          [zadd_left_commute,zadd_commute,zadd_assoc]
                          delsimps [inj_zint])));
by(res_inst_tac [("s","(k + #1) + $# z"), ("t","k + ($# z + #1)")] subst 1);
be (signature_log_mono_trace RS subsetD) 2; ba 3;
  by(asm_full_simp_tac (simpset() addsimps [in_naturals]) 2);
  by(ALLGOALS(asm_full_simp_tac (simpset() addsimps
                          [zadd_left_commute,zadd_commute,zadd_assoc]
                          delsimps [inj_zint])));
qed "HSD_3";
```

References

[1] D. Basin. A formal analysis of secure pc. Internal Paper.

[2] A. D. Brucker and B. Wolff. A case study of a formalized security architecture. In *Electronic Notes in Theoretical Computer Science*, volume 80. Elsevier Science Publishers, 2003.

[3] Unknown. The specification of the hysteresis signature system using darma. Internal Paper.

Index