Analysis and Implementation of Secure Electronic Mail Protocols

Milica Barjaktarovic (Wilkes University), Shiu-Kai Chin (Syracuse University/Rome Lab), John Faust (Rome Lab), Chet Hosmer (Odyssey Research Associates), David Rosenthal (Odyssey Research Associates), Geoffrey Hird (Odyssey Research Associates), Maureen Stillman (Odyssey Research Associates), and Dan Zhou (Syracuse University)

Note: This work supported by USAF Rome Laboratory, NSA and Air Force Office of Scientific Research

1 Introduction

An ultimate goal of research in high-confidence systems is the development of theories, technologies, and practices which will routinely assure the correctness, safety and security of systems. This paper defines a formal design process by which we are able to both synthesize and verify a secure electronic mail system built from hardware and software components.

Over the past year, a team comprised of E-Systems, ORA, Rome Lab, NSA, Syracuse University, and Wilkes University has been working on formal descriptions, analysis, and implementations of a secure electronic mail system. The center of activity was the Multilevel Information System Security Initiative (MISSI) described below. In addition, we investigated an electronic mail application running on top of MISSI called the Defense Messaging System (DMS). In order to analyze the lower level details of MISSI, we investigated Privacy Enhanced Mail (PEM), which is similar to MISSI.

The Multilevel Information System Security Initiative (MISSI) is a National Security Agency (NSA) program, designed to send protected messages over unprotected networks such as Internet [1]. MISSI provides the following security services: data integrity and authentication (via hashing, digital signatures, and certificate authentication); confidentiality (via public and secret cryptography); non-repudiation with proof of origin; and non-repudiation with proof of receipt (optional). Users' public cryptography information is posted in the public distributed database called the Directory, in files called certificates [2], [7]. Users are authorized by certification authorities (CAs), which issue certificates ("tickets") to the users.

MISSI-capable workstations are equipped with either FORTEZZA Cards (F) or FORTEZZA Plus cards (FP), or they have no cards. A FORTEZZA card is a credit card sized Personal Computer Memory Card Interface Association (PCMCIA) card, used to perform security services such as encryption. F cards handle Sensitive but Unclassified (SBU) messages, and FP cards handle Secret (S) messages. Users perform MISSI functions using software called User Agent (UA), which interfaces with the software and hardware of the Card. Each user can have several "personalities" of possibly different security levels.

MISSI is a more complex version of Privacy Enhanced Mail (PEM). MISSI relies on several standards: X.500 for Directory services, X.509 for certificate management, FORTEZZA and MSP SDN.701 for security services, and X.400 suite or RFC822/MIME for message transport (Fig. 1). If X.400 is used, ACP 123 and its Annexes must be used.

PROTOCOL FUNCTION	IMPLEMENTATION
1. message preparation	e-mail program (e.g. emacs)
2. message content format	User Agent (UA) based on MISSI, and X.420 or RFC822
3. security services	MSP protocol, based on SDN.701
4. message transport	X.411 or SMTP(MIME)

Fig.1: MISSI message preparation architecture

At the lower layers, we are implementing the Privacy Enhanced Mail (PEM) protocol. PEM, as described in RFC 1421, is a simpler version of MISSI. It supports both public and private key cryptography, authentication using X.509 certificates, and integrity checking using signed hash functions.

The DMS is a layer built on top of the MISSI architecture that implements an e-mail system which handles messages at a variety of levels of trust.

The work of this group included formalizing requirements, architecture and design. We see this as a model of collaboration on efforts that involve industry adoption of formal techniques. This transfer has been coordinated between the universities, government, and commercial industry. As a collaboration team, the group has been able to make greater progress on the formalization of the MISSI program elements than any individual team member could expect to accomplish alone. This paper provides a description of

the progress of the group in the specific areas of: 1) problems surfaced by working with E-Systems on the Defense Messaging System (DMS) and the formalization of DMS requirements; and 2) the development of a formal design process for hardware and software applied to Privacy Enhanced Mail (PEM) which is similar to MISSI.

Section 2 discusses work on the analysis of DMS requirements. Section 3 presents a model of MISSI using Promela. Section 4 is a discussion of the hardware and software design process as applied to Privacy Enhanced Mail (PEM). At the end of each section is a discussion of the impact we believe that the work has made on the requirements, design and implementation processes.

2 Analysis of DMS Requirements

2.1 Introduction

In this section, we describe ORA's effort in DMS requirements analysis. The purpose of this effort was to develop an approach for modeling a system that used MISSI security technology. Due to current commercial interest, we chose to examine a part of the Defense Message System (DMS), called the User Agent (UA). Our goal was to formulate a modeling method of the UA requirements, which could aid system engineers in uncovering potential ambiguities and inconsistencies in the informal requirements. Additionally, we wanted to demonstrate the effectiveness of the method by showing where some of the DMS requirement descriptions needed to be augmented.

E-Systems, a DMS developer, provided ORA with a pared down version of the set of requirements that were relevant for this effort. Since even this list of requirements was large and the semantic concepts that they covered were diverse, we needed to develop an approach that could aid in surfacing potential problems, but would not entail the cost and complexity of a comprehensive model. ORA developed a mixed approach that integrated the informal specification with a partially formalized specification. We constructed a simplified state machine model with one temporal-logic construct in order to express important aspects of the requirements. Various underlying mathematical constructs that were not directly relevant were left at an informal level.

2.2 Modeling Method

The DMS requirements refer to properties of a large and complex system. Consider the following requirement.

acp.123.202.ac Originator Indication

Within the MMHS, when the originator field is present, it should include the directory name (if one is available) in the formal-name sub-field (auto fill-in). The free-form-name or telephone-number sub-fields may also be present. This service shall be supported. When this indication is present it must be displayed to the user. (See X.400 B.55)

An attempt to mathematically model this requirement fully, would include not only higher-level information such as whether a service is implemented, but lower-level information about fields and subfields, the user's capability to interact with them, etc. This would be a considerable undertaking and the results would be complex. Moreover, these entities are often not mentioned elsewhere in other requirements, and so it would be of little use to model them for surfacing potential requirement conflicts. In addition, the added complexity would reduce the likelihood practitioners would read and benefit from the specification.

The first design decision was to identify high level information in the requirements, with the emphasis on concepts that were common to many of the requirements. The main aspects of the requirements that were initially examined were whether a UA service is supported, and how a message indication should be displayed. Although indications are usually displayable, there are a variety of ways in which this can occur. For example, they may be displayable in a regular fashion or they may need to be displayed "prominently."

The second design decision was to specify the model in a syntactic form that was extremely simple. We used notational conventions familiar from mathematics and programming languages, and we kept the formal logic down to a minimum. The underlying structures were a state machine and messages. In the current version of the model, the emphasis has been on simplicity, even if it means that some of the specification details are suppressed. This aids in the accessibility of the presentation, but has the potential for leaving some

ambiguities unresolved.

The specification begins with some of the essential aspects of the UA state machine and then the individual requirements are modeled. Of course, it would be useful to have as full a definition of the UA as possible, in order to capture completely all of the details that might be mentioned in the requirements. However, the presentation is simplified by initially avoiding a laborious explanation of aspects of the UA that are not needed for most requirements. Where needed in the requirements, additional constructs were inserted in the specification.

The individual requirements for DMS have considerable history (having evolved through various organizations and documents), and engineers would like to continue to base their work on these individual statements. Thus, we chose to present the model in a largely 1-1 correspondence with the requirements.

2.3 Example

We illustrate the modeling approach by presenting the form of a sample requirement.

REQUIREMENT (from E-Systems)

acp.123.201.j Typed body - acp123

If the type of the body is important to provide context for the message, it must be displayed to the user.

```
FORMAL SPECIFICATION DESCRIPTION

Describe mode and content of display.  Attempt to describe when
display is required.

FORMAL SPECIFICATION COMMENTS

need_body_type_for_context(msg) is the function that returns True
when the reader of msg needs to know the body type to understand it.

There may be multiple body parts.  The terms body_type and
need_body_type_for_context, used in the specification below,
have a hidden index for selecting a particular body part.
```

```
FORMAL SPECIFICATION

CONSTANT UNKNOWN

AXIOM 1

  need_body_type_for_context(MSG)
=>
  UA.display_mode.body_type = regular
  AND UA.display_content.body_type = MSG.body_type

AXIOM 2

need_body_type_for_context(MSG) = UNKNOWN

FORMAL SPECIFICATION ISSUES  (unresolved)

a) What criteria does the UA use to determine whether the
reader needs to see the body type of a message?
```

2.4 Impact

The result of this effort is the development of an approach for modeling a large set of requirements with a diverse semantic content. The level of formality introduced, although not comprehensive, was effective at surfacing ambiguities in the specification. The modeling method also maintained a close correspondence with the original informal requirements.

Ideally, this type of modeling and analysis of the requirements would be performed once, instead of by each developer of a commercial system that needs to meet these requirements.

This work is in progress and the final results will be included in a report that will be delivered to the Air Force.

3 Modeling MISSI using Promela

In this section, we describe a Wilkes University effort in modeling MISSI. We formally defined various requirements either directly or indirectly implied in the standards and documents describing MISSI, such as message structure and MISSI access and forwarding policy. We wrote a specification for MISSI sender, receiver, forwarding, local cache, and a Secure Network Server (SNS).

We chose formal language PROMELA, based on Hoare's CSP, to specify and test various MISSI components. PROMELA is a language based on Hoare's CSP and implemented in C. SPIN is the publicly distributed automated verification tool which supports PROMELA [12], [14], [13], [22].

PROMELA uses processes that can be thought of as C functions executed concurrently. PROMELA uses channels for communication. A channel is a FIFO queue of a certain capacity. The following syntax is used to append a message type `msg_type` with values `expr1` and `expr2` to channel `q`: `q!msg_type(expr1, expr2)`. `q?msg` means "receive message `msg` on channel `q`."

SPIN can use `assert` statements, `never` claims and Linear Temporal Logic (LTL) statements for validation. Assert statements state properties of data at a given state. Never claims state properties of data at one or more states, and can be used to model LTL formulae.

3.1 Model

We modeled a MISSI system consisting of sender and receiver workstations, which are on separate LANs connected via a lossless WAN. Each LAN includes a Secure Network Server (SNS), which functions as a guard/firewall. A MISSI Local Area Network (LAN) is called an "enclave." As an outbound filter, i.e. when checking messages which are leaving the (Secret) enclave, SNS will pass messages according to the following criteria: all Secret (S) messages must be encrypted; all messages must be signed. SNS will also check if the message originator is allowed to send messages with security level lower than the enclave security level. As an inbound filter, i.e. when checking messages which are entering the enclave, SNS will accept messages according to one of the following three policies, messages must be signed; or messages must be encrypted; or any messages are accepted.

In order to fully utilize model checkers, it is necessary to have a manageable number of states, because a model checker keeps track of all possible states and performs analysis on this state table. There are several ways to reduce number of states: to bundle several consecutive execution steps in one unit using `atomic` statements; to reduce channel sizes, counters, assert and other statements, and number of variables in general; and to make an abstraction of the system and focus on the areas of interest only.

We employed all these approaches in our modeling. Our model consists of modules, where we abstract the modules which are less important by their input-output function, and focus on the modules of interest.

Currently, we have three different models of MISSI: [3] specifies a detailed model of MISSI sender with an abstraction of local cache and certificate verification; [4] also specifies the sender, but focuses on the local cache and certificate verification, and abstracts away the processes necessary to prepare a message for sending; [5] model abstracts away the sending process, and focuses on the receiver.

3.2 Results

In order to formalize MISSI access policy, we put together a more formal definition of terms and rules scattered throughout MISSI documentation. We also proposed solutions in case of ambiguity, conflict, or incomplete specifications.

For example, we define the term "enclave security level" and specify the enclave requirement as: "Enclave security level is equal to the highest security level of the workstations which form the enclave. An enclave can contain workstations of lower security level. Workstation security level equals the maximum security level that the workstation is capable of processing, e.g. Secret if the workstation has FORTEZZA Plus card reader."

We formalized the local access policy as follows: "Any personality of security level X is allowed to send email to any other personality of security level Y, if: the messages sent are classified at the security level Z, where $Z \leq \min(X,Y)$; and the messages sent have security service W applied.

Security service W applied to outgoing messages is determined as:

1. We assume that SBU enclaves are allowed to send unprotected messages to the WAN.

2. All messages classified as Secret must be signed and encrypted; i.e. W=signed+encrypted if msg_level==S. Secret enclaves cannot send unprotected messages.

3. Secret enclaves cannot receive unprotected messages. That is, W≥signed if wkst_level==S. In other words, Secret enclaves must send and receive signed or signed-and-encrypted messages. This requirement is enforced by SNS filtering [21], p.11.

4. Vanilla workstations can receive only unprotected messages, because they do not have any FORTEZZA capability.

5. The above discussion brings us to formalize one additional requirement: Secret enclaves can communicate with vanilla workstations only if the vanilla workstations are on an enclave with SNS. Currently, only Secret enclaves are obliged to have SNS.

We ran our SPIN model on an Indigo-2 with 320Mb of RAM. The models took several seconds to execute. In the following paragraphs we list sample properties we proved.

If we forward a message, the resulting message classification must be greater that the individual message parts; and the resulting message must be organizational if the original was organizational.

It is necessary to prohibit non-organizational users from forwarding organizational messages. Therefore, we propose the following requirement: *Every message sent must have the following property: If a message M consists of message text T and several body parts B1, B2, ... Bn, the message M is assigned organizational status iff any of the body parts has organizational status and it is possible to assign organizational status to M. If it is not possible to assign organizational status to M, the message cannot be sent.*

In order to preserve the requirement that certification paths stored in the local cashe are "unbroken", we have to delete all certificates and CRLs below an invalid one. This solution was missing from the local cashe specification.

MSP functions are called in the proper order, e.g. `msp_status` after `msp_login`.

3.3 Impact

Since various layers in the MISSI protocol stack need to interface with each other, MISSI designers and implementors are faced with a multidimensional

problem: first, MISSI system by itself is relatively complex, taking into account FORTEZZA cards, cryptography, authentication, and integrity checking; and second, MISSI must interface with the layer above it and the layer below it, as well as to certificate management and manipulation standards. There are many issues when interfacing two or more (independent) standards, because the requirements might not be specified precisely, or they might conflict. For example, MISSI messages which travel using X.400 must be sent as military messages (MMs). MMs are specified in ACP123 and ACP123 Annex A. MMs can be individual or organizational. Organizational messages commit organizations, and require special security concerns. For example, if MISSI user A sends an organizational military message to MISSI user B, can B forward it to a non-organizational user?

In our experience, English specifications are ambiguous and incomplete. Even specifications that sound correct can lead to incorrect results. At minimum, English specifications should contain some kind of pseudo-code or flow diagram, in order to see more clearly all "if-then-else" branchings and other possibilities. Formal specifications and proofs, augmented with textual descriptions, provide a very clear and detailed documentation for the system. This aids in understanding of the system, and exposes weak areas of system design. Formal specification can be used as a blueprint for implementation, and formal proofs can be used as a blueprint for implementation testing.

Formal verification assures of desired system properties. Desired system properties consist of original specification, enhanced by additional requirements that formal specification uncovers.

SPIN is useful as a tool, especially since it resembles C. Model checkers are easy to learn and use, and they provide quick specification and testing ability with the focus on the control path. The tradeoff is that model checkers can run out of memory in case of detailed data analysis and/or very large models. The best approach is to abstract away all unnecessary details and/or divide a model into several sub-models and validate each sub-model separately. The sub-models can represent the same system, where each sub-model emphasizes a particular feature of the system and simplifies the other features (e.g. one sub-model focuses on login and logout, another on sending mail, another on receiving mail). The sub-models can be parts of a bigger model (e.g. each submodel is a layer in a layered protocol.)

4 Assured Implementation

In this section we describe an ongoing effort at Syracuse University to create a highly-assured implementation of Privacy Enhanced Mail (PEM) using a combination of theorem-provers, model checkers, formal specification refinement tools, and hardware compilers. One purpose of the effort is to demonstrate the practicality of a formal design process for high-confidence hardware and software.

The work described here builds on two previous efforts to formally model NSA's Multilevel Information System Security Initiative (MISSI). The first effort by Johnson, Saydjari, and Van Tassel in [8] defines various MISSI security properties in higher-order logic. The MISSI Certificate Authority Workstation (CAW) has been modeled by Marron using *PROMELA* and the *SPIN* model checker, [18].

MISSI is based in part on Privacy Enhanced Mail (PEM). PEM is described in four *Request for Comment* (RFC) papers: RFC 1421, RFC 1422, RFC 1423, and RFC 1424, [17]. While the message field names and structure may differ somewhat between MISSI and PEM, the formal design process used here is applicable to both.

The design process used here is an extension of the design method described in the DIMACS Network Threats workshop [9]. In [9], PEM message structures and operations were defined and verified against top-level security properties such as privacy, authentication, and integrity.

We have 1)formally defined the security properties of interest, namely privacy, authenticity, non-repudiation, and integrity; 2) formally verified that the secure message structures and associated operations on them are correct and secure; and 3) generated C++ code from the formal descriptions which is correct by construction. We have used a variety of formal methods tools including the HOL90 theorem prover from the University of Combridge, the Spin model-checker from ATT, and the Specware specification refinement system and code generator from Kestrel Institute. The effort for formalization, synthesis and verification, while sometimes intricate, is well within the capabilities of engineers who have formal methods training.

Section 4.1 gives an overview of the high-confidence design process. Section 4.2 describes the design and verification of the top-level PEM processes such as the sender and receiver processes. Section 4.3 describes how C++ is generated from verified specification descriptions. Section 4.4 gives an

overview of the hardware design process.

4.1 Design Process Overview

The specification and implementation of PEM is divided into two parts – the *control path* and the *data path*. The control path in PEM corresponds to the sender and receiver processes. The data path in PEM contains values derived from actual message structures (e.g. *ENCRYPTED, MIC_CLEAR, and MIC_ONLY*), and operations on messages such as encryption, hashing, and integrity checking.

The control path is described using a process algebra and verified by a model checker. Data path objects such as messages are described in higher-order logic along with various operations on messages. These operations are shown to satisfy the desired security properties.

Message structures, and the operations on them, are implemented in both hardware and software. Cryptographic algorithms such as encryption and hashing are done in hardware. Other message operations, such as integrity checking, are implemented in C++ which have been obtained by a specification refinement process followed by code generation.

Both hardware and software operations on messages are described and verified in higher-order logic. Message structures are defined as abstract data types. Doing so leads to straightforward definitions of functions for message creation and interpretation.

Security operations such as integrity checking are defined using encryption and hash algorithms as parameters and message destructor functions to access parameter values such as specific hash function names, message integrity codes, etc. The correctness of these operations is verified using higher-order logic (see [9]). C++ code for these operations is obtained by specification refinement and code generation [23].

The actual implementation of hash and encryption functions is done using Xilinx field programmable gate arrays (FPGA) and/or custom VLSI circuits. The functions and implementations are described and verified in higher-order logic. FPGA netlists and VLSI layouts are obtained by compiling the implementation descriptions in higher-order logic into hardware generation functions within a VLSI CAD system [15].

4.2 Sender and Receiver Processes

PEM sender and receiver processes are implemented in C++ which are specified using a CSP (Communicating Sequential Process)-like [11] process language called *PROMELA* and the *SPIN* model checker, [13]. The sender and receiver processes correspond to the "control path" of a process. They specify the functions called and the sequence of the function calls. For example, at the top level, the receiver process does the following: 1) determines the message type, i.e., *ENCRYPTED, MIC_CLEAR, MIC_ONLY*, etc.; 2) performs authentication based on the certificates included in the message header; 3) decrypts and/or decodes the message text; and 4) performs an integrity check by comparing the hash of the received message to the message integrity code (MIC) contained in the message header.

At this level, functions (such as the integrity checking function), are treated as "black boxes" which take certain inputs and produce certain outputs. The details of what takes place inside the "black boxes" are described, verified, and synthesized elsewhere. The correctness of the functions is assumed at this level. Function calls correspond to ports or channels in the process description.

Modal properties of the sender and receiver processes are verified using SPIN. Properties such as *it is always the case that the integrity checking function is called for all message types* are verified.

For example, all MIC_CLEAR messages must be checked for integrity. We model function MIC_CLEAR_is_Intact as a "black box" which takes a message and produces a Boolean response stating if the message is intact or not. `d1,d2,d3,d4,d5,d6,d7,d8,d9,d10` and `d11`

are fields in PEM message header, and `d12` is the data portion of the message.

```
MIC_CLEAR_is_Intact[wkst]!d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12;
from_MIC_CLEAR_is_Intact[wkst]?rsp;
if
:: (rsp == OK) -> skip;                    /*proceed,message is intact*/
:: (rsp == FAIL) -> goto error_processing; /*based on the error policy,
                                             take appropriate action*/
fi;
```

The function MIC_CLEAR_is_Intact is a *port* in the above process description. The result of the integrity check is a value `rsp` returned by

MIC_CLEAR_is_Intact. If `rsp` is false, i.e., if the message is not intact then an error has occurred. If `rsp` is true, then the message is intact.

The correctness and implementation of MIC_CLEAR_is_Intact is shown in the following section.

4.3 Software

To illustrate how security functions are implemented with assurance, we examine how message integrity is checked for MIC_CLEAR messages, i.e., for messages which are transmitted in the clear but include a message integrity code (MIC).

`is_Intact` is defined for message integrity checking. It takes several parameters: 1)*verify* - a function verifies the signature, which takes a plain text message, a signature and a key, and returns *true* if the signature is signed on the given plain text with the private key paired with the given key, otherwise, it returns *false.* 2) *hash* - the message digest algorithm; 3) *message* - the plain text part of the message retrieved from the mail; 4) *ekey* - the public key of originator used by the recipient to verify a signature; and 5) *mic* - the received digital signature of the message. It declares both the message and its digital signature are intact if the verification of the digital signature of the original message against the hash of the received message succeeds.

```
is_Intact
    ⊢def  ∀verify hash message mic ekey.
          is_Intact verify hash message mic ekey =
          verify (hash message) mic ekey
```

The assumptions made about the received message are: 1) the received signature is generated by signing the hash (message digest) of the transmitted message; 2) it is computationally infeasible to find two messages m_1 and m_2 which hash to the same value, so if two hashes are equal the two messages are the same; and 3) the verification process succeeds if-and-only-if the signature is generated on the plain text that is being verified.

What we want is for `is_Intact` to be true is-and-only-if the received message is identical to the one transmitted. Under these assumptions, the correctness theorem is proved using the definition of `is_Intact` with the assumed properties in the antecedent of the nested implication.

```
is_Intact_msg
    ⊢  ∀verify sign hash txmessage rxmessage txmic rxmic ekey dkey.
         (txmic = sign (hash txmessage) dkey) ⊃
         (rxmic = txmic) ⊃
         (∀m1 m2. (hash m1 = hash m2) ⊃  (m1 = m2)) ⊃
         (∀s1 s2. verify s1 (sign s2 dkey) ekey = s1 = s2) ⊃
         ((rxmessage = txmessage)
           = is_Intact verify hash rxmessage rxmic ekey)
```

The definition and correctness theorem for `is_Intact` show that given the proper parameters, message integrity will be checked correctly. For each type of PEM message, the parameters needed by `is_Intact` must be extracted from the various message fields.

PEM messages are thought of as *tuples*. Each kind of PEM message is defined as a data type in higher-order logic with accessor functions to obtain the parameters within the message header fields. By defining PEM messages as types, we can eliminate tuples which never occur which simplifies verification. How types are introduced in higher-order logic is described by Melham in [19].

`MIC_CLEAR_is_Intact` shown below is the integrity checking function for MIC_CLEAR messages. It has as part of its definition the use of message accessor functions to get the parameter values needed by `is_Intact`.

$$
\begin{array}{l}
MIC_CLEAR_is_Intact\ msg = \\
\quad (let\ micInfo = get_MIC_CLEAR_MIC_Info\ msg \\
\quad in \\
\quad\ let\ ekey = \\
\quad\ get_Key_from_ID\ (get_OriginatorAsymID_info\ mic_clear_msg) \\
\quad in \\
\quad\ is_Intact \\
\quad\ (MIC_sign_select\ micInfo) \\
\quad\ (MIC_hash_select\ micInfo) \\
\quad\ (get_MIC_CLEAR_text\ msg) \\
\quad\ (get_MIC_mic\ micInfo)\ ekey)
\end{array}
\tag{1}
$$

The correctness of `MIC_CLEAR_is_Intact` is declared by the following theorem.

$$
\begin{aligned}
&\vdash \forall mic_clear_msg\ sign\ txmessage\ dkey. \\
&\quad let\ micInfo = \\
&\quad\ get_MIC_CLEAR_MIC_Info\ mic_clear_msg \\
&\quad in \\
&\quad let\ ekey = get_Key_from_ID \\
&\quad\ (get_Originator_AsymID_info\ mic_clear_msg) \\
&\quad in \\
&\quad let\ hash = MIC_hash_select\ micInfo \\
&\quad\ and \\
&\quad verify = MIC_sign_select\ micInfo \\
&\quad\ and \\
&\quad rxmessage = get_MIC_CLEAR_text\ mic_clear_msg \\
&\quad in \\
&\quad (get_MIC_mic\ micInfo = sign\ (hash\ txmessage)\ dkey) \\
&\quad\ \supset \\
&\quad (\forall m1\ m2.(hash\ m1 = \ hash\ m2) \supset (m1 = m2)) \supset \\
&\quad (\forall m1\ m2.verify\ m1\ (sign\ m2\ dkey)\ ekey = m1 = m2) \supset \\
&\quad ((txmessage = rxmessage) = \\
&\quad\ MIC_CLEAR_is_Intact\ mic_clear_msg)
\end{aligned}
\tag{2}
$$

The above correctness theorem states that `MIC_CLEAR_is_Intact` is true if-and-only-if the transmitted and received messages are the same provided the hash and signature verification functions gotten from the message header have the properties of being one-to-one and invertible, respectively.

With each kind of message formally defined as a data type with verified security functions, the types and functions are refined into code by the Specware specification refinement system [16]. Specware supports both composition of specifications and refinements (implementations) using category and sheaf theory [23]. It depends heavily on refining abstract data types to concrete representations. The correctness of these representations is checked using the HOL theorem-prover. When the Specware refinement uses only existing data types and operations in its C++ library, code can be synthesized. The language of Specware is *Slang* [16]. Slang is Specware's *specification language* and roughly corresponds to the higher-order logic in the HOL system

[10]. It is straightforward to map HOL terms into Slang.

4.4 Hardware

The cryptographic functions such as DES, RSA, MD–2, and MD–5 are implemented in hardware using an assured hardware design process. The assured hardware implementation process in is described in COMPASS '97 [6].

The process uses a link between structural descriptions in higher-order and HOL, and the Mentor Graphics GDT VLSI design system [20]. A compiler called HOL2GDT [15] maps recursive and non-recursive structural descriptions in HOL to parameterized cell generators in GDT. These cell generators are called to create layouts and net lists for custom chips. The GDT net lists can be compiled into Xilinx net list format for FPGA implementations. The HOL2GDT compiler requires that every hardware component in HOL have a corresponding cell generator in GDT. New components are introduced into the GDT and HOL libraries by creating new cell generators from HOL definitions.

The hardware design process is as follows. HOL structural descriptions are compiled into GDT and simulated. Simulations can be multi-level so both logical and electrical simulations are possible. Simulation usually catches most of the errors. After the initial simulations are done and initial design corrections are made, formal verification is done. This catches the subtle errors.

This process has been used to create many designs, both custom VLSI circuits and FPGA implementations. [6] describes a working bit-serial multiplier chip which was fabricated at MOSIS. Our intent is to use the same process to implement cryptographic functions such as DES, RSA, MD–2, and MD–5 in hardware.

4.5 Impact

The conclusions we draw from our implementation experience thus far are as follows:

- The use of several formal methods and tools is practical.
- While at times intricate and detailed, the proofs of security properties as they relate to actual message structures are not conceptually

complicated and are within the capabilities of engineers with proper training.

- Combining verification with hardware and software synthesis is possible and desirable.

In short, the high-confidence design process we used is promising and appears to be practical on design problems of meaningful size and scope.

5 Conclusions

The collaborative team of E-Systems, ORA, Rome Lab, NSA, Syracuse University, and Wilkes University has done a formalization of selected portions of DMS requirements, MISSI and PEM from formal specification to implementation over the last year. The formal approach has added greater clarity to the specification and assurance to the implementation.

Our conclusions are:

- The formal specification, verification and synthesis of secure message structures and operations is straightforward.
- The formal design process reveals precisely the necessary properties of components in order for the system to operate correctly and securely.
- The formal design process is a practical one which produces a high-confidence implementation. The results to date are promising in that they show the practicality of coupling formal analysis with formal synthesis of assured hardware and software.
- Even without the use of formal methods, there is a critical need for more precise informal specifications, as the MISSI and DMS requirements allowed for too many interpretations. By adding formalism to more precise english language specifications, we can further increase precision.

References

[1] *FORTEZZA Application Developers Guide.* Publicly available documents at
http://www.armadillo.huntsville.al.us, 14 July 1995.

[2] National Security Agency and SDNS Vendor Participants. Secure Data Network System SDNS Message Security Protocol (MSP). SDN.701 Revision 3.0, 21 March 1984.

[3] Milica Barjaktarović. Formal Specification and Verification of MISSI Architecture Using SPIN. Technical Report 4, AFOSR Summer Research Program, Rome Laboratory, Rome NY, September 1996.

[4] Milica Barjaktarović. Formal Specification and Verification of MISSI Local Cashe Using SPIN. Technical report, Rome Laboratory and ORA, Rome Laboratory, Rome NY, December 1996.

[5] Milica Barjaktarović. Formal Specification and Verification of MISSI Reciever and Forwarding Policy Using SPIN. Technical report, AFOSR Summer Research Program, Rome Laboratory, Rome NY, May 1997.

[6] Milica Barjaktarović. Formal Specification and Verification of MISSI Sender Using SPIN. In *COMPASS97*, submitted November 1996.

[7] CCITT. Data Communication Networks Message Handling Systems, Recommendations x.500-x.521. Technical report.

[8] John P. Van Tassel D. Randolf Johnson, Fay F. Saydjari. MISSI Security Policy: A Formal Approach. Technical Report R2SPO-TR001-95, INFOSEC Research and Technology Group, National Security Agency, July 1995.

[9] Dan Zhou and Shiu-Kai Chin. Verifying Privacy Enhanced Mail Functions with Higher Order Logic. In *DIMACS Workshop on Network Threats*, Rutgers University, Piscataway, NJ, December 4–6 1996.

[10] M.J.C. Gordon. A proof generating system for higher-order logic. In G. Birtwistle and P. A. Subramanyam, editors, *VLSI specification, verification and synthesis.* Kluwer, 1987.

[11] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, New York, 1985.

[12] Gerard J. Holtzman. *Validation and Verification of Communication Protocols*. Prentice Hall, New York, 1989.

[13] Gerard J. Holzmann. Basic SPIN Manual. AT&T Bell Laboratories, Murray Hill, New Jersey, 079074.

[14] Gerard J. Holzmann. Design and Validation of Protocols: a Tutorial. *Computer Networks and ISDN Systems*, 25:981–1017, 1993.

[15] Juin-Yeu Lu and Shiu-Kai Chin. Linking HOL to a VLSI CAD System. In Jeffrey J. Joyce and Carl-Johan H. Seger, editors, *Higher Order Logic Theorem Proving and Its Applications*, number 780 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[16] Kestrel Institute, 3260 Hillview Ave, Palo Alto, CA. *Specware Language Manual*, 1.02 edition, June 1995.

[17] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC 1421, DEC, February 1993. ftp: ds.internic.net.

[18] Christopher S. Marron. A PROMELA Model of the MISSI Architecture. Technical Report R2SPO-TR002-95, INFOSEC Research and Technology Group, National Security Agency, 21 September 1995.

[19] T. Melham. Automating Recursive Type Definitions in Higher Order Logic. In G. Birtwistle and P. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 341–386. Springer-Verlag, 1989.

[20] Mentor Graphics Inc., San Jose, CA. *GDT Led, Lx Standard Cell, Explorer Lsim V.5.3 users manuals*, 1990.

[21] National Security Agency, http://www.armadillo.huntsville.al.us. *Network Security Managers (NSM) Functional Requirements Specification and Concept of Operations (CONOP)*, June 1996.

[22] J.M. Troya P. Merino. Modeling and Verification of the ITU-T Multipoint Communication Service with SPIN. In *Proceedings of the 2nd International Workshop on the SPIN Verification System*, DIMACS, Rutgers University, New Brunswick, New Jersey, August 1996.

[23] Yellamraju V. Srinivas and Richard Jullig. Specware: Formal Support for Composing Software. In *Conference on Mathematics of Program Construction*, Kloster Irsee, Germany, July 1995.