A Mechanized Logic for Secure Key Escrow Protocol Verification *

Tom Schubert and Sarah Mocas

Department of Computer Science
Portland State University

Abstract. Reasoning about key escrow protocols has increasingly become an important issue. The Escrowed Encryption Standard (EES) has been proposed as a US government standard for the encryption of unclassified telecommunications. One unique feature of this system is key escrow. The purpose of key escrow is to allow government access to session keys shared by EES devices. We develop a framework to formally specify and verify the correctness of key escrow protocols that we mechanize within the HOL theorem proving system. Our logic closely follows the logic, SVO , used for analyzing cryptographic protocols which was developed by Syverson and vanOorschot [13]. Using the HOL mechanization of SVO , we formally demonstrate the failure of the EES key escrow system by showing that it does not insure that the escrow agent receives correct information. This was previously shown experimentally [2]. Last, we offer an alternative escrow protocol and demonstrate its correctness.

1 Introduction

Several logics for analyzing cryptographic protocols and authentication schemes have been proposed (see for example [1, 3, 8, 13, 14]). The primary goal of this type of analysis is to verify the security of a protocol or authentication scheme thereby insuring that an unauthorized third party does not have access to secret information that is exchanged between the users of a protocol. In this paper, we develop a logic for analyzing key escrow systems. In considering a key escrow system, the goal is different in that we want to insure that a third party can decode encrypted information sent between two parties, hence insuring the "integrity" of the escrow agent.

Our logic for analyzing key escrow systems closely follows the logic used for analyzing protocols that was developed by Syverson and vanOorschot [13]. Syverson and vanOorschot's logic was chosen as it encompasses many of the desirable features from earlier developed logics in an integrated approach. Using our mechanization of SVO, we formally demonstrate the failure of the EES key escrow system to insure that the escrow agent receives correct information. This was previously shown experimentally [2]. We also present an alternative escrow protocol and demonstrate its correctness.

* Submitted to the 1995 International Workshop on Higher Order Logic and its Applications

In this paper, we will first describe key escrow protocols and the EES key escrow mechanism. Then, we will present our mechanization of a belief logic based on the SVO logic and the extensions required to reason about key escrow protocols. After presenting examples of key escrow protocol verification, we will conclude with a brief description of future work.

1.1 Background

Reasoning about key escrow protocols has increasingly become an important issue. The Escrowed Encryption Standard (EES) defines a group of US Government cryptographic chips including both the Clipper and Capstone chips [7]. EES has been proposed as a US government standard for sensitive, but unclassified government and civilian telecommunications. One unusual and highly controversial feature of this system of encryption is key escrow. Key escrow is achieved as follows. Each EES device is assigned a unique identifier and secret key during the manufacturing process. This identifier and key are then stored in escrow. The initial communication for any encrypted session between two EES processors involves the transmission of a "Law Enforcement Access Field" (LEAF), which contains the unique identifier and an encrypted copy (encrypted with the escrowed device specific key) of a previously agreeded upon session key. The purpose of this transmission is to allow legal government access to the session key generally via wiretapping. Many of the details of the Clipper chip and escrow scheme are classified, but much of the information that is available can be found in [2, 5].

Although the EES cryptographic algorithm "Skipjack," used by both the Clipper and Capstone chips, is classified, many of the details of the escrow scheme are public and open to scrutiny. Consequently flaws in this scheme have been detected. M. Blaze has outlined various techniques that allow secure communication between two users of EES devices without transmission of a valid LEAF. We reiterate some of his observations and subject them to more formal analysis. Most importantly, Blaze showed that there is a reasonable scenario under which a dishonest user can interoperate with an honest user without sending a correct LEAF.

Two models of cheating will be considered. In one model, two users of EES devices, A and B, cheat (intentionally preventing the escrow agent from receiving a correct session key). In the second model one user, A, is dishonest and user B is honest.

In the first model it is trivial for A and B to use EES devices without giving the escrow agent a valid key. For example, A can simply apply some other form of shared cryptography, such as exclusive-or with a random string (one-time pad) or encoding using a public-key system, to the LEAF when it is sent. Upon receiving the modified LEAF, B can reverse this process prior to passing the LEAF to its EES device. This may seem like an unreasonable action in that A and B could instead use their alternate, shared cryptosystem to communicate all of their messages but in fact their solution is very practical. Both Clipper and Capstone are designed for fast encryption/decryption of large quantities of information, whereas many alternative cryptosystems are slow or expensive.

The more interesting case to consider consists of one dishonest user, A, and an honest user, B. An ideal key escrow system would, in this case, insure that either the escrow agent has received a valid session key and chip identifier or that the honest user would not continue communication. Using our mechanization of the SVO logic, we can show that, given the available information on the data contained in the LEAF, this guarantee is not implicit in the government scheme. Further, we propose a modification to the LEAF that insures that, with high probability, either the escrow agent has received a valid session key and chip identifier or that the honest user will not continue communication. We verify that these properties hold for this modification using HOL.

1.2 Key Escrow

The key escrow mechanism used by EES devices is dependent on the transmission of the encoded session key and user identification. As stated above, both of these pieces of information are passed in one transmission in the LEAF. It is this transmission that we will focus on. The LEAF consists of 128 bits that are divided as follows:

UID	32 bit unique unit identifier (the serial number of the chip),
$\{SK\}_{UK}$	80 bit session key, SK, encoded using the device unit key, UK,
CS	16 bit LEAF checksum.

The information described above is then encoded using a fixed family key, FK, resulting in the LEAF, $\{UID,\ \{SK\}_{UK}\ ,\ CS\}_{FK}$. Session keys are exchanged between two users prior to the transmission of the LEAF. The LEAF transmission is the direct result of initializing an EES device for a session by entering the session key.

An initialization vector, IV, is also transmitted with the LEAF and it appears that the calculation of the checksum depends on both the session key and the IV. M. Blaze observed that all bits in the LEAF change when the IV or session key changes [2]. The exact construction of the CS and IV are not public.

During the manufacturing process each EES device is assigned a unique identifier UID and unit key UK. Further, groups of devices are assigned a single family key, FK. The unit key is constructed from the exclusive-or of two other keys, $KU1$ and $KU2$ that are then each stored separately with the UID by two designated government agencies, the escrow agents. For further detail see D. Denning's paper [5].

On intercepting a valid LEAF, a government agent may then decrypt the LEAF with the family key, use the UID to retrieve the keys $KU1$ and $KU2$ from the escrow agents, and then exclusive-or these two keys to obtain the unit key KU. As each LEAF contains a session key encoded with KU the government agent may now obtain the session key for two users of EES devices.

2 Formal Belief Logic

Belief logics are designed to consider what conclusions individual parties (principals) in a communication dialog can deduce based on messages received and a set of initial assumptions or beliefs. Logics devised to reason about cryptographic protocols generally consider only idealized protocols; there are no bit streams, but rather typed messages. Thus, all parties are presumed to recognize varying message formats, despite the lack of this information in the bit stream. Belief analysis attempts to show that only desired properties are guaranteed by the communication (data security, non-repudiation, no replayed transactions, etc). Note that proofs about idealized protocols are not a guarantee that the concrete protocols are correct. There are many implementation assumptions that if invalid, would cause a secure, idealized protocol to actually be insecure. For example, these logics all assume that the cryptoalgorithm is secure.

An interesting aspect to the SVO logic is its use of "abstracted protocols." SVO messages may also include propositions about held beliefs. For example, if a principal A were to pass a message to principal B that included a new encryption key that A believed to be a good secret key, the logic permits the message to include not just the key, but also A's (implicit) belief that it is a good key.

Using the HOL system type definition mechanisms, we define a number of application specific data types. HOL's recursive type definition facility [9] automates the process of defining new data types in terms of already existing types. Both new type constants and type constructors (operators) can be defined. Additional (recursive) functions can be defined to operate on the concrete data representation of the type. The properties of new types must be derived by formal proof. This guarantees that the type does not introduce inconsistency into the logic.

2.1 Types

To model protocols where possibly encrypted messages are passed among different principles, we construct new data types for principals, keys, and message items. Principals are easily defined as unique entities and will exhibit independent behavior and hold autonomous beliefs.

```
let PRINCIPAL = define_type `principal` `principal = PRINCIPAL num`;;
```

There are two different types of cryptographic algorithms (symmetric key and public key) and several ways in which a cryptosystem and key may be used (data encryption, digital signature, key exchange agreement)[2]. To distinguish between these variations, types **key** and **keyuse** are created.

[2] A introduction to cryptographic algorithms may be found in [6] or [10].

```
new_type_abbrev( `crypt`, ":bool");;
define( "PUBLIC  = T");;
define( "PRIVATE = F");;

let KEY = define_type `key` `key = NONE | SECRET num | PUB crypt num`;;
let KEYUSE = define_type `keyuse` `keyuse = DATA | SIG | XCHG`;;
```

Messages consist of items that may be names of principals, keys, data, and nonces (an indicator of the timeliness of a message). Recall that the abstract protocol also allows assertions to be passed in messages. For example, the protocol permits a principal to indicate (**CLAIM**) its confidence in a public or shared secret key. Additionally, the abstract protocol allows messages to be signed without indicating how this is achieved. We have also added the possibility that a LEAF field may be sent (described in section 1.2). To simplify the LEAF specific inference rules (described in the following section), we add to the LEAF definition an indicator that shows which principal is the intended recipient. Since the exact construction of the CS field has not been publicly disclosed, we define this field as a **:num**.

```
let SIGNED = define_type `signed` `signed =  UNSIGNED
                                           | SIGNED principal`;;

let KEYFACT = define_type `keyFact` `keyfact =
                            PK    principal keyuse key
                          | SHARE principal principal key`;;

let ITEM = define_type `item` `item = EMPTY
                        | INFO  num
                        | NAME  principal
                        | KEY   key
                        | CLAIM  keyfact
                        | NONCE num
                        | LEAF  principal principal (key#key) num`;;

new_type_abbrev(`message`, ":key# signed # (item)list");;
```

2.2 Propositions

The SVO logic defines a number of property predicates to describe the set of beliefs present in a system. This set may also include "facts" that may not be believed by all principals. Informally[3], a message can be *received*, but it may be encrypted. If a principal *has* the key, then the plaintext may be obtained. However, the principal has no assurance who the plaintext came from unless the message is signed. If a message is signed, then a principal can assume that the signing principal *said* the plaintext. But we don't know when the message was said, unless it is *fresh*, in which case, the principal can assume the signer effectively, *says* the message at the same time it is received (the message is timely,

[3] A discussion on the rational for these constructors is beyond the scope of this paper.

and thus not a replay). Still, the message may not come from an authority (e.g. "Simon says") unless the signer *controls* the message. If so, then whatever the message *asserts* is considered accurate.

```
let PROPOSITION = define_type `prop`
    `prop =   ASSERT   keyfact
            | BELIEVES principal prop
            | CONTROLS principal keyfact
            | SEES     principal (item)list
            | SAYS     principal (item)list
            | RECEIVED principal message
            | SAID     principal (item)list
            | FRESH    (item)list
            | HAS      principal key`;;
```

The SVO axiom schemata, with some small additions and omissions, is given next.

Rules and Axioms: The SVO logic has two primitive inference rules (modus ponens and necessitation) and a significant axiom schemata for reasoning about protocols. These rules are captured in HOL by an inductively defined relation [4], **INFER**. The necessitation rule states that anything derivable from axioms alone can be believed by a principal.

Believing: Principals may believe a proposition if it logically follows from already held beliefs.

P *believes* $\varphi \ \wedge\ \mathsf{P}$ *believes* $(\varphi \supset \psi) \supset \mathsf{P}$ *believes* ψ

P *believes* $\varphi \supset \mathsf{P}$ *believes* (P *believes* φ)

Receiving: If a principal has received an unencrypted message, then she has received the concatenates of the message. If a principal has received an encrypted message and has the decryption key, the principal has effectively received a unencrypted copy of the message.

P *received* $(X_1, ..., X_n) \supset \mathsf{P}\, received X_i$

P *received* $\{X\}_k \ \wedge\ \mathsf{P}$ *has* $\hat{K} \supset \mathsf{P}$ *received* X

Seeing: If a principal receives an unencrypted message, then she sees all the concatenates and any function (encryption or decryption) that can be applied to the concatenates.

P *received* $X \supset \mathsf{P}$ *sees* X

P *sees* $(X_1, ...X_n) \supset \mathsf{P}$ *sees* X_i

P *sees* $(X_1) \ \wedge\ ... \ \wedge\ \mathsf{P}$ *sees* $(X_n) \supset \mathsf{P}$ *sees* $F(X_1, ...X_n)$

Having: If a principal sees a key, then the principal has the key and vice-versa (note only keys can be "had").

P *has* $k \equiv \mathsf{P}$ *sees* k

Source Association: Principals can deduce the identity of the sender of a message if the message is encrypted with the senders signature public key or if the message is signed using a shared secret key.

$\mathsf{P} \overset{k}{\leftrightarrow} \mathsf{Q} \ \wedge \ \mathsf{R}\ \textit{received}\ \{X^Q\}_k \supset \mathsf{Q}\ \textit{said}\ X$
$PK_\sigma(\mathsf{Q}, K) \ \wedge \ \mathsf{R}\ \textit{received}\ \{X\}_{k-1} \ \wedge \ \mathsf{P}\ \textit{received}\ \{X^Q\}_k \supset \mathsf{Q}\ \textit{said}\ X$

Saying: A principal sees anything that she said.
$\mathsf{P}\ \textit{said}\ (X_1, ...X_n) \supset (\mathsf{P}\ \textit{said}\ X_i \rightarrow \mathsf{P}\ \textit{sees}\ X_i)$
$\mathsf{P}\ \textit{says}\ (X_1, ...X_n) \supset (\mathsf{P}\ \textit{said}\ (X_1, ...X_n) \ \wedge \ \mathsf{P}\ \textit{says}\ X_i)$

Freshness: To insure that protocols are not susceptible to replay attacks, only recent (fresh) messages are considered valid. A message is fresh if any concatenate is fresh and encrypted fresh messages are also fresh.
$\textit{fresh}\,(X_i) \supset \textit{fresh}\,(X_1, ...X_n)$
$\textit{fresh}\,(X_1, ...X_n) \supset \textit{fresh}\,(F(X_1, ...X_n))$

Nonce-Verification: If a received message is fresh can be attributed to a particular principal, then we can infer that the principal "says" the message.
$(\textit{fresh}\,(X) \ \wedge \ \mathsf{P}\ \textit{said}\ X) \supset \mathsf{P}\ \textit{says}\ X$

Jurisdiction: If a principal is a trusted authority on a property, then anything she says can be taken as true.
$(\mathsf{P}\ \textit{controls}\ \varphi \ \wedge \ \mathsf{P}\ \textit{says}\ \varphi) \supset \varphi$

Symmetric goodness of shared keys: If two principals share a secret key, then:
$\mathsf{P} \overset{k}{\leftrightarrow} \mathsf{Q} \equiv \mathsf{Q} \overset{k}{\leftrightarrow} \mathsf{P}$

EES LEAF Validation: If a leaf is received and the receiving principal has the secret keys, then the receiver may believe that the claimed assertion is true.

$$(\mathsf{E}\ \textit{received}\ \{LEAF\ \mathsf{P}\,\mathsf{Q}\ \{sk\}_k\ n\}_{fk} \ \wedge \ \mathsf{E}\ \textit{has}\ \hat{k} \ \wedge \ \mathsf{E}\ \textit{has}\ \hat{fk})$$
$$\supset \mathsf{E}\ \textit{believes}\ \mathsf{P} \overset{sk}{\leftrightarrow} \mathsf{Q}$$

Note this rule is not in the original SVO logic and, in fact, we will show that it leads to undesirable conclusions. Using the SVO logic, we might state this rule as an assumption made by the escrow agent. A natural way to define principal specific inference rules might be to add an additional proposition type constructor that expected a list of propositions and a new proposition that could be drawn from the list of propositions. In our mechanization, we chose to defer the resulting recursive type problem to future work when we port our mechanization to HOL90.

SVO also defines rules for "Key Agreement" and "Comprehending," which we have chosen not to include.

$$PK_\sigma(\mathsf{P}, K_P) \ \wedge \ PK_\sigma(\mathsf{Q}, K_Q) \supset \mathsf{P} \overset{K_{PQ}}{\leftrightarrow} \mathsf{Q}$$
$$\mathsf{P}\ \textit{believes}\ (\mathsf{P}\ \textit{sees}\ F(X)) \supset \mathsf{P}\ \textit{believes}\ (\mathsf{P}\ \textit{sees}\ X)$$
$$\mathsf{P}\ \textit{received}\ F(X) \ \wedge \ \mathsf{P}\ \textit{believes}\ (\mathsf{P}\ \textit{sees}\ X) \supset \mathsf{P}\ \textit{believes}\ \mathsf{P}\ \textit{received}\ F(X)$$

For our application, key agreement occurs external to the escrow protocol and thus, the rule is never used. The comprehending rules allow a principal to *see* a function of her inputs. This function can be any one-one function and in practice, is encryption or decryption. Since these functions can already be represented in the logic, it is unnecessary to provide a second form of representation.

2.3 Tactics and Lemmas

Performing proofs using only the above axioms is exceedingly tedious. We initially developed a collection of simple tactics to simplify proof manipulations. These tactics were used to prove a number of general lemmas that form the basis for derived inference rules. A more sophisticated tactic mechanism will be described in the next section.

In the box below, Lemma0 states that if a principal has a key and has received a message encrypted with that key, then the principal can infer he has received the message. Lemma1 states that if a key is part of a received, but encrypted message, the new key can be obtained if the receiving principal has the message decryption key. Lemma3 states that if a principal has the key for a received, encrypted message, then the principal can obtain any component of the plaintext.

```
Lemma0
  ⊢ ∀ P k s L l. RECEIVED P (k,s,l) isIN L  ∧
                 HAS P (DKEY k) isIN L ⟹
                 INFER L (RECEIVED P (NONE,s,l))
Lemma1
  ⊢ ∀ l A B fk sk s L. (KEY sk) subItem l  ∧
      (RECEIVED B(fk,s,l)) isIN L  ∧  (HAS B(DKEY fk)) isIN L ⟹
      INFER L(HAS B sk)

Lemma2
  ⊢ ∀ l x A B fk s L.
      x subItem l  ∧  (RECEIVED B(fk,s,l)) isIN L  ∧
      (HAS B(DKEY fk)) isIN L ⟹
      INFER L(RECEIVED B(NONE,s,[x]))
```

The lemmas below simplify the process of showing that a principal can believe a received claim when the claim is part of a source authenticated message from an appropriate authority.

```
Lemma3
  ⊢ ∀ l n k A B L cl.
      (NONCE n) subItem l              ∧  (CLAIM cl) subItem l    ∧
      (ASSERT(SHARE A B k)) isIN L      ∧  (CONTROLS A cl) isIN L  ∧
      (RECEIVED B(k,SIGNED A,l)) isIN L  ∧  (HAS B(DKEY k)) isIN L ⟹
      INFER L(ASSERT cl)

Lemma4
  ⊢ ∀ l n k s A B L cl.
      (NONCE n) subItem l         ∧  (CLAIM cl) subItem l  ∧
      (RECEIVED B(k,s,l)) isIN L  ∧  (ASSERT(PK A SIG k)) isIN L  ∧
      (HAS B(DKEY k)) isIN L      ∧  (CONTROLS A cl) isIN L  ⟹
      INFER L(ASSERT cl)

Lemma5
  ⊢ ∀ l n k s A B L cl.
       NONCE n subItem l          ∧  CLAIM cl subItem l  ∧
       RECEIVED B (k,s,l) isIN L  ∧  ASSERT (PK A SIG k) isIN L  ∧
       HAS B (DKEY k) isIN L      ∧  CONTROLS A cl isIN L ⟹
       INFER L (ASSERT cl)
```

Using the mechanization developed above, it is straightforward to prove a number of simple inferences. For example, to show that an escrow agent believes it obtains the correct secret key that is passed between two principals A and B (where A and B are EES devices), requires a simple specialization of lemma4 above.

```
⊢ ∀ A B E fk sk n.
  let L =
     [HAS B(DKEY fk);BELIEVES E(HAS E(DKEY fk));ASSERT(SHARE A E fk);
      ASSERT(SHARE A B fk);CONTROLS A(SHARE A B sk);
      RECEIVED E(fk,SIGNED A,[NONCE n;CLAIM(SHARE A B sk)])]
  in
   INFER L(BELIEVES E(ASSERT(SHARE A B sk)))
```

2.4 General Support for Reasoning about Inductive Definitions

To better support the use of the mechanized SVO logic, we have developed a general infrastructure for creating automated and interactive procedures to prove goals about inductively defined relations. The infrastructure provides tools to build interactive functions and goal-directed support functions, procedures to eliminate existentially quantified variables from terms, and tactics to generalize existentially quantified rules. Many of the functions are generic, expecting as arguments the list of inference rules returned by `new_inductive_definition` and a list of the relevant combinators. In our case, the list of relevant combinators includes all of the type constructors from the newly defined types (`key, keyuse, keyfact, item,` and `proposition`).

We note that for many applications, validation of the inductively defined rules is, at least initially, of greater importance than automated proofs. For the automated proofs to be useful, their construction must be understood. Thus, it is valuable for the automated procedures to inform the user what rules are relevant and when rules are applied to a goal.

Interactive support is provided by a collection of parameterized functions that suggest an appropriate rule to apply based on the current goal or a term. (`suggested_rules()` and `rules_for_term()`, respectively). Using these mechanisms, users don't need to remember all the rules that may apply, but can request a list of potentially useful rules. Many protocol logics have a significant number of inference rules ([8] lists 44 rules). Even SVO logic, which was designed to significantly reduce the number of rules compared with other protocol logics, requires 23 rules.

We have also developed a tactic generating function, `make_suggest_tac` that returns a tactic specialized for a list of inference rules and a list of combinators. The tactic searches for and then applies, an appropriate rule to apply based on the current goal. In practice, this tactic appears to provide a significant improvement over tactics that try every rule. The tactic also outputs what rules where chosen to apply in a given situation. The displayed list of decisions can assist the user understand why a proof is correct and often, why it fails.

A second general tactic EXISTS_ELIM_TAC has also been developed to search for witnesses to replace existentially quantified variables. The tactic searches the assumption list for candidates that match the variable's usage in the goal. After replacement, goals can often be reduced by rewriting with the assumption list. For performance considerations, the tactic defers rewriting until all existentially quantified variables have been replaced.

Using the above two tactics, a general strategy was developed and programmed into a tactic (IND_RULES_TAC). The initial proofs of the lemmas required a fair number of steps, but with the use of the IND_RULES_TAC, the proofs are performed with only a few tactics. The resulting proofs also require less time with fewer intermediate theorems generated. At this point in development, the tactic does not backtrack and so occasionally, the user must select the correct rule.

The final general support mechanism developed supports generalizing some of the rules returned by new_inductive_definition. When creating a new inductive definition, there are often rules with existentially quantified variables in the antecedent that could instead be universally quantified. While this is not always beneficial, in many cases, proofs are simplified.. For example, the first Seeing rule states that a principal receives an unencrypted message, the principal also *sees* the message.

$$\vdash \forall \mathtt{L\,P\,X}.(\exists \mathtt{s}.\mathtt{INFER\,L(RECEIVED\,P(NONE,s,X))}) \Longrightarrow \mathtt{INFER\,L(SEES\,P\,X)}$$

Note the variable s is a placeholder for the signed message field and can be converted to a universally quantified variable. The general (forward proof) function can convert this rule to:

$$\vdash \forall \mathtt{L\,P\,X\,s}.\mathtt{INFER\,L(RECEIVED\,P(NONE,s,X))} \Longrightarrow \mathtt{INFER\,L(SEES\,P\,X)}$$

3 Key Escrow Protocol Verification

This section will describe how the logic developed in the previous section can be used to reason about the EES key escrow protocols. In modeling the proposed government protocol, the principles used in this section are:

A, B	two users,
E	a government agent (loosely referred to as the escrow agent),
C_A, C_B	the EES chips used by A and B, respectively.

The principals C_A and C_B have been included so that it is clear that users, not encryption devices, have the potential for dishonest actions. Under certain scenarios, we rely on the "honesty" of the EES devices to insure the integrity of the messages received through the protocol. We have intentionally not included C_E, the escrow agent's EES technology. The key escrow protocol is strictly for the benefit of the escrow agent and so we can assume that E is honest.

Protocol analysis typically occurs at an abstract level, making a fair number of assumptions about the properties of the cryptographic system underlying the protocols. In practice, cryptographic systems do not necessarily guarantee the assumptions that are made by protocol analysis tools[12]. Below we state our set of assumptions. Ideally, we would like to represent the concrete cryptographic system details in HOL to validate these assumptions.

1. $\mathsf{C_A}$ and $\mathsf{C_B}$ are tamper resistant and not faulty,
2. A and B have previously used a secure method to exchange the session key SK,
3. The cryptographic algorithms are computationally secure,
4. The transmission between an EES device, $\mathsf{C_X}$, and the owner of the device, X, is secure.

Initially we will use a simpler model consisting of the principals:

M	a potentially dishonest agent,
E	a government agent (loosely referred to as the escrow agent),
$\mathsf{C_A}$	an EES chip,
$\mathsf{C_B}$	an EES chip.

In this case we will add the following assumptions about the initialization of the EES devices:

1. On behalf of $\mathsf{C_A}$ and $\mathsf{C_B}$ a secure method to exchange the session key SK has been executed,
2. $\mathsf{C_A}$ and $\mathsf{C_B}$ have both been given SK.

In all examples, the message that is passed is referred to as the LEAF. In the government key escrow system, the initialization vector, IV, is passed with the LEAF. We have omitted explicit reference to the IV as it is passed in the clear and any cryptographic function that it serves is not publicly known.

3.1 Government Key Escrow

As previously stated, ideally we want to insure that the escrow agent receives a valid chip identifier, UID, and encoded session key, $\{SK\}_{UK}$, from which SK can be retrieved. With respect to our logic, this translates into showing that the escrow agent, E, believes that she has a valid UID and encoded session key. That is, E *believes* (E *sees* UID) and E *believes* (E *sees* $\{SK\}_{UK}$) where UID and $\{SK\}_{UK}$ are from a valid LEAF and SK is the session key that is shared by the EES devices.

Using our simplified model,where A and B are omitted, the basic escrow protocol consists of the following message:

$$\text{Message 1} \quad \mathsf{C_A} \rightarrow \mathsf{C_B}, \mathsf{E} : \text{LEAF}$$

indicating that $\mathsf{C_A}$ sends $\mathsf{C_B}$ and E a LEAF. The EES protocol expects the following relationships to exist prior to transfer of the LEAF.

- The sharing of the unit keys is established when the EES devices are manufactured. We also assume E recognizes the device identifiers UID_A, UID_B.

$\mathsf{E} \overset{UK_A}{\leftrightarrow} \mathsf{C_A}$ and $\mathsf{E} \overset{UK_B}{\leftrightarrow} \mathsf{C_B}$

E *believes* $(\mathsf{E} \overset{UK_A}{\leftrightarrow} \mathsf{C_A})$ and $\mathsf{C_A}$ *believes* $(\mathsf{E} \overset{UK_A}{\leftrightarrow} \mathsf{C_A})$

E *believes* $(\mathsf{E} \overset{UK_B}{\leftrightarrow} \mathsf{C_B})$ and $\mathsf{C_B}$ *believes* $(\mathsf{E} \overset{UK_B}{\leftrightarrow} \mathsf{C_B})$

– $\mathsf{C_A}$ and $\mathsf{C_B}$ have a secure session key.

$\mathsf{C_A} \overset{SK}{\leftrightarrow} \mathsf{C_B}$

$\mathsf{C_A}$ *believes* $(\mathsf{C_A} \overset{SK}{\leftrightarrow} \mathsf{C_B})$ and $\mathsf{C_B}$ *believes* $(\mathsf{C_A} \overset{SK}{\leftrightarrow} \mathsf{C_B})$

– EES devices control the messages that they generate.

$\mathsf{C_A}$ *controls* LEAF and $\mathsf{C_B}$ *controls* LEAF

– We assume that each principal that *shares* a key, *has* the shared key.

The LEAF is constructed by $\mathsf{C_A}$ (which by (P *controls* $\varphi \wedge$ P *says* $\varphi) \supset \varphi$ gives the LEAF) and the transfer of the LEAF takes place as follows:

$\mathsf{C_A}$ *said* LEAF
$\mathsf{C_B}$ *received* LEAF
E *received* LEAF

Note the idealization process required by SVO and other logics results in a loss of information about the order in which messages are sent and received. To avoid this step and retain sequence information, a more sophisticated representation could be achieved by using a process algebra notation.

Consider the conclusions that can be drawn by the escrow agent E. Any principal who receives a LEAF and has the LEAF (symmetric) family key can also see the contents of the LEAF. Given $\mathsf{C_B}$ and E have the family key and applying the second Receiving rule, both principals then also receive $(UID_{C_A}, \{SK\}_{UK-C_A}, CS)$, the list of components of the LEAF. By the first Seeing rule, $\mathsf{C_B}$ and E also *see* the list. Further applying the second Seeing rule, $\mathsf{C_B}$ and E see the components $UID_{C_A}, \{SK\}_{UK-C_A}$ and CS.

Since E sees both UID_{C_A} and $\{SK\}_{UK-C_A}$ and has $(UID_{C_A}, UK - C_A)$ in escrow, then by the second Receiving rule, we can conclude that E sees the session key SK and that E believes that the session key that he has is the same session key that was sent by $\mathsf{C_A}$ and shared between $\mathsf{C_A}$ and $\mathsf{C_B}$. Using our mechanization in HOL, a proof of this behavior can be easily constructed.

```
⊢ ∀ A B E fk sk ka n s.
  let L =
        [HAS E(DKEY ka);HAS E(DKEY fk);ASSERT(SHARE A E ka);
         ASSERT(SHARE A E fk);ASSERT(SHARE A B sk);
         RECEIVED E(fk,s,[LEAF A B(ka,sk)n])]
  in
   INFER L(BELIEVES E(ASSERT(SHARE A B sk)))
```

Unfortunately, as M. Blaze has shown, it may not be the same session key that $\mathsf{C_A}$ and $\mathsf{C_B}$ will eventually use. We will use the word *correct* to refer to the LEAF that was said by $\mathsf{C_A}$.

Consider the following modification of our simplified protocol where a malicious principal, M, intercepts and modifies the LEAF.

Message 1	$\mathsf{C_A} \rightarrow \mathsf{M}$	:LEAF
Message 2	$\mathsf{M} \rightarrow \mathsf{C_B}, \mathsf{E}$	:LEAF$'$

Here the message LEAF$'$, is not the message sent by $\mathsf{C_A}$, but is sufficiently similar so that $\mathsf{C_B}$ accepts LEAF$'$ and continues to interoperate with $\mathsf{C_A}$. This is represented in SVO as:

$\mathsf{C_A}$ *said* LEAF
M *received* LEAF
M *said* LEAF$'$
$\mathsf{C_B}$ *received* LEAF$'$
E *received* LEAF$'$

Using the same series of steps given above, we can conclude that E sees a session key SK' for user UID' (retrieved from LEAF$'$). Consequently, if E believes that LEAF is a message that was sent from $\mathsf{C_A}$, then E believes that LEAF$'$ is also a message that was sent from $\mathsf{C_A}$. The substitution of LEAF$'$ for LEAF is justified by the experimental work done by Blaze.

The government key escrow protocol extends the number of principals to include A and B, the owners of the EES devices.

Message 1	$\mathsf{C_A} \rightarrow \mathsf{A}$	: LEAF
Message 2	$\mathsf{A} \rightarrow \mathsf{B}, \mathsf{E}$	: LEAF
Message 3	$\mathsf{B} \rightarrow \mathsf{C_B}$	: LEAF

Letting A play the role of M, our HOL mechanization allows us to prove that the escrow agent E fails to obtain the shared key. Thus, the government protocol fails for one honest user and one dishonest user.

```
⊢ ∀ A B E fk sk sk' ka n s.
   let L =
         [HAS E(DKEY ka);ASSERT(SHARE A E ka);HAS E(DKEY fk);
          ASSERT(SHARE A E fk);BELIEVES B(ASSERT(SHARE A B sk));
          BELIEVES A(ASSERT(SHARE A B sk));
          RECEIVED E(fk,s,[LEAF A B(ka,sk')n])]
   in
    (INFER L(BELIEVES E(ASSERT(SHARE A B sk')))  ∧
     INFER L(BELIEVES B(ASSERT(SHARE A B sk)))  ∧
     INFER L(BELIEVES A(ASSERT(SHARE A B sk))))
```

M. Blaze showed that any randomly generated 128 bit string will have a $1/2^{16}$ chance of appearing valid for the current session key and IV. An attack that replaces a correct LEAF with a bogus LEAF appears to be feasible in practice [2]. We will therefore assume that an EES device does not currently use the checksum to validate the contents of the LEAF to an extent that would allow E to believe that the correct LEAF had been transmitted.

We consider two additional assumptions each of which can be used to convince E that she had received the correct information.

1. E *believes* (B *believes* (B *received* CORRECT_LEAF)).
2. E *believes* ($\mathsf{C_B}$ *believes* ($\mathsf{C_B}$ *received* CORRECT_LEAF)).

The first assumption assumes that B has the computational resources to either decode a message encoded with the family key and then determine that the message is correct or to determine that it is correct without using the family key. In either case, these are extremely strong assumptions. The second assumption does offer a solution that will be outlined in the next section. The rule E *believes* (A *said* CORRECT_LEAF) should not convince E of the correctness of the LEAF since there is no guarantee that the LEAF that E received is the LEAF that A sent. It appears that E must use $\mathsf{C_B}$ to verify the protocol.

We note though, given the government protocol, not only can we not conclude that $\mathsf{C_B}$ believes that $\mathsf{C_A}$'s UID and encoded session key were transmitted we can not even show that $\mathsf{C_B}$ believes that the LEAF that he received was sent by $\mathsf{C_A}$ only that it was sent by some EES device. This is not a surprising result as M. Blaze has already shown experimentally that a dishonest user can send an honest user an invalid LEAF that will be accepted as correct.

3.2 Modified Government Key Escrow

We propose that the LEAF be modified to include an additional field that contains an encrypted version of the UID and encrypted session key. The LEAF is now:

$$\{UID,\ \{SK\}_{UK}\,,\ \{\,\{SK\}_{UK}\,,\ UID\}_{SK}\,,\ CS\}_{FK}\,.$$

We also propose that any so modified EES device that receives a LEAF check that:

$$\text{if } (\,\{SK\}_{UK}{}',\ UID') = D_{SK}(\,\{SK\}_{UK}\,,\ UID)$$

$$\text{then } \{SK\}_{UK}{}' = \{SK\}_{UK} \ \text{ and } UID' = UID.$$

Further, if these equalities do not hold, then we require that the device terminate the session. Given these modifications we argue that E believes that she has a valid UID and encoded session key. In fact, E's belief is solely based on the belief that an honest user, in this case $\mathsf{C_B}$, believes that a valid LEAF has been transmitted and received. This is the second assumption listed above, namely E *believes* ($\mathsf{C_B}$ *believes* ($\mathsf{C_B}$ *received* CORRECT_LEAF)).

In this case, $\mathsf{C_B}$ *believes principal*($\mathsf{C_B}$ *received* CORRECT_LEAF) since only $\mathsf{C_A}$ and $\mathsf{C_B}$ share SK and $\{SK\}_{UK}{}' = \{SK\}_{UK}$ and $UID' = UID$ where $(\,\{SK\}_{UK}{}',\ UID') = D_{SK}(\,\{\,\{SK\}_{UK}\,,\ UID\}_{SK}\,)$ The modification adds a level of authentication for $\mathsf{C_A}$ that appears (given known information) to be missing from the government EES protocol. This modification is enough to verify, with high probability, that the LEAF came from $\mathsf{C_A}$ and has not been modified.

The mechanization of this protocol and the authentication of $\mathsf{C_A}$ in HOL can be accomplished by modifying the type definition for items so that the LEAF type constructor also includes the new authentication field to represent the encryption of the $\{\{SK\}_{UK}\,,\ UID\}_{SK}$. The LEAF authentication proof rule must also then be modified to check that the authentication field is correct.

It is fairly obvious that a message using a small hash function to validate the message content is weak (collisions are easy to find). Our modification is stronger than simply using a larger hash function since additional authentication is provided for the sender through the use of a shared key. Device C_{B} can now verify that the LEAF is from C_{A}, since only C_{B} and C_{A} share the "hash function" E_{sk}. It should be noted that our modification does increase the length of the LEAF by 112 bits. Even though key exchange occurs only once during a session, this may complicate a hardware implementation. We are unsure as to whether or not the checksum field can be removed. Though our modification replaces the known use of this field, its interaction with the IV field is not public.

4 Conclusions

In this paper we have described the embedding of the SVO logic within the HOL theorem proving system. The logic has been extended to support reasoning about key escrow protocols and we have demonstrated its use with the original EES key escrow protocol and an improved protocol. As a byproduct of this development, we have also created an infrastructure to support proofs using inductively defined relations.

As observed by Syverson and vanOorschot[13], the SVO logic is still a beginning step in the development of a logic that reasons about cryptographic protocols. Likewise, our adaptation of SVO as a tool to reason about escrow protocols and specifically our mechanization of this logic in HOL are only initial steps. Our current mechanization only allows propositions about keys to be passed as part of the abstract protocol. We would like to define propositions as functions of items passed in messages and also permit messages to include propositions. We expect that when porting to HOL90, the mutually recursive data types can be more easily defined.

We also plan on further developing semi-automated decision procedures. Many of the tactics developed greatly assisted in simplifying the proof development process, but it is clear that the tactics can be more fully automated.

Finally, we intend to extend the mechanization so that the behavior of principals are defined as within a process algebra. This work will be based on earlier work by one of the authors[11]. Analysis using SVO and other belief logics constructs the set of messages sent and beliefs by hand as part of the *protocol idealization* step. However, the behavior of a system can be modeled as a value passing process algebra where *agents* can perform *actions* send, receive, and generate. Put in this framework, the process of creating the belief set from principal definitions could be automated. This will allow us to model networks where messages are broadcast, private networks, and malicious principals that replace part of the network (*man-in-the-middle attacks*).

References

1. M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM Press, 1991.
2. Matt Blaze. Protocol failure in the escrow encryption standard. In Lance J. Hoffman, editor, *Building in Big Brother*. Springer-Verlag, 1995.
3. Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1), Feb. 1990.
4. Juanito Camilleri and Tom Melham. Reasoning with inductively defined relations in the HOL theorem prover. Technical Report 265, University of Cambridge Computer Laboratory, 1992.
5. D. Denning. The U.S. key escrow encrytion technology. *Computer Communications*, to appear.
6. Dorothy Denning. *Cryptography and Data Security*. Addision-Wesley, 1982.
7. National Institute for Standards and Technology. Escrow encrytion standard. *Federal Information Processing Standards Pubulication 185*, 1994.
8. L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1990.
9. Tom Melham. Automating recursive type definitions in higher order logic. In G. Birtwhistle and P.A Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 341–386. Springer-Verlag, 1989.
10. Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc, 1994.
11. E. Thomas Schubert. A hybrid model for reasoning about composed hardware systems. *Conference on Computer-Aided Verification*, June 1994.
12. Gustavus J. Simmons. Cryptanalysis and protocol failures. *Communications of the ACM*, 37(11), November 1994.
13. Paul F. Syverson and Paul C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, May 1994.
14. P. vanOorschot. Extending cryptographic logics of cryptographic logics of belief to key agreement protocols. In *Proceedings of the First ACM Conference on Computers and Communications Security*, pages 232–243, 1993.

This article was processed using the LaTeX macro package with LLNCS style