1999

Formal Analysis of a Secure Communication Channel: Secure Core-Email Protocol

Dan Zhou
Syracuse University

Shiu-Kai Chin
Syracuse University, chin@cat.syr.edu

Formal Analysis of a Secure Communication Channel: Secure Core-Email Protocol

Dan Zhou and Shiu-Kai Chin

Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, New York, 13244
{danzhou, chin}@cat.syr.edu

Abstract. To construct a highly-assured implementation of secure communication channels we must have clear definitions of the security services, the channels, and under what assumptions these channels provide the desired services. We formally define secure channel services and develop a detailed example. The example is a core protocol common to a family of secure email systems. We identify the necessary properties of cryptographic algorithms to ensure that the email protocol is secure, and we verify that the email protocol provides secure services under these assumptions. We carry out the definitions and verifications in higher-order logic using the HOL theorem-prover. All our definitions and theorems are conservative extensions to the logic of HOL.

1 Introduction

Numerous security protocols are used for secure transactions in networked systems. To construct high-confidence implementations of these protocols, we need to have protocols that provide security services and to implement them correctly. One way of establish the correctness of protocols is to model, specify and verify them in higher-order logic. We demonstrate how this can be done in this paper.

Protocols such as Kerberos [14] and Needham-Schroeder [13] authentication protocols are based on message exchanges between two or more parties. In general, these protocols and the logics (such as [3]) and tools (e.g., [10]) analyzing them have assumed that a single message passing between two parties is secure if the message is appropriately encrypted and signed and if the keys for decryption and signing are kept secret. In this work, we explore the validity of this assumption by studying secure communication channels. We identify what it means for a channel to be secure and the required properties of cryptographic functions to ensure channel security.

We have two goals. First, we want precise definitions of the services desired of secure channels. Some applications require a channel with integrity where messages cannot be modified without detection. Other applications require a channel that is confidential, where only the intended recipient can read the message. We formalize these secure protections in higher-order logic as properties that secure channels should satisfy.

Second, we want clear definitions of the required properties of cryptographic algorithms used in secure channels. As we use cryptographic algorithms in protocols to

provide secure communication, the properties of these algorithms are vital in reasoning about the security properties of the secure channels. The required properties vary, depending on the particular services the channels provide and the components of the channels themselves. As an example, we formally specify a secure core-email protocol that provides confidentiality, integrity, source authentication, and non-repudiation. The protocol uses a combination of secret-key encryption, public-key encryption and digital signatures. It is common to a family of secure email systems such as Privacy Enhanced Mail (PEM) [9] and Pretty Good Privacy (PGP) [16].

We identify and specify the properties required of cryptographic algorithms for the channel to be secure. The secure core-email protocol is then verified formally to provide secure services under these assumptions. The list of required properties can serve as a reference when specific algorithms are used in actual protocol implementations.

The purpose of our work is not to invent new protocols. Rather we want to add enough formality to the protocol analysis so that we can account for security properties in concrete implementations. As a practical demonstration, we have carried out the formal development process down to the generation of C++ code of the secure core-email protocol and Privacy Enhanced Email and have reported the result in [15] . This paper concentrates on a formal analysis of the secure channel.

Our work attempts to fill the gap between previous abstract formal treatments such as Lampson and others [8], and detailed implementation descriptions such as PEM,[9]. The focus of abstract analysis in [8] is how to make secure decisions based on user statements. The correct functionality of secure channels is assumed. The focus of detailed implementation descriptions is on message structure and protocols. Definitions of security properties are missing and no attempt is made to show the protocols and operations on messages satisfy the intended security properties. This paper attempts to relate concrete implementations to abstract security properties.

There are two types of methods of analyzing protocols. There are those based on theorem proving and those based on model checking. In the category of theorem proving, specialized logics are developed to describe both protocols and their desired properties, inference rules are defined to reason about the correctness of protocols. For example, BAN logic [3] and authentication logic by Lapmson and others [8] are used for describe and reason about authentication protocols. Brackin has embedded an extension of GNY logic (called BGNY logic) in higher-order logic theorem prover HOL and has developed specialized tactic in HOL to prove theorems about protocols [2]. By embedding BGNY logic in HOL as a conservative (definitional) extension, his analyzer has advantage of the mechanized theorem proving environment and guarantees the correctness of the theorems. In comparison, our work uses general higher-order logic and relies on the higher-order logic itself for specification and reasoning. Higher-order logic has been used in constructing assured implementation of computer systems [4]. Those specialized logics are more abstract than higher-order logic which our work employed. It is not clear how we can arrive at a correct implementation from protocols described in these logic without translating the protocols descriptions to a language that is closer to implementation.

In the category of model checking, protocols are described as state machines, properties are expressed either as invariants or as another state machine. NRL protocol ana-

lyzer uses first-order logic to express invariants and searches the state space (potentially exhaustively) to find if the invariants hold for the protocol [11]. Spi-calculus models both protocols and desired properties as traces and uses equivalence of processes to reason about the correctness of protocols [1]. NRL protocol analyzer provides the automation of analysis, spi-calculus is suitable for modeling concurrent systems. However they are all further away from constructing an assured implementation than ours.

For this study we use the higher-order logic theorem prover HOL [5] for formal specification and verification. We use standard predicate calculus notation. The symbols $\wedge, \vee, \neg$, and $\supset$, respectively, denote the logic operations *and, or, negation*, and *implication*, while $\forall$ and $\exists$ denote *universal* and *existential quantifications*. Function composition is denoted by the symbol $\circ$, and fa denotes the application of function f to a. The symbol I denotes the identity function. Expression $\Gamma \vdash t$ denotes a theorem: whenever the list of logical terms in Γ are all *true*, the conclusion t is guaranteed to be *true*. Definitional extensions to the HOL system are denoted by $\vdash_{def}$.

For the rest of the paper we start by describing rigorously the cryptographic algorithms and their properties in Section 2. This is followed by the formal definitions of services of secure communication channels in Section 3. In Section 4 we present an example channel that is a secure core-email protocol common to a family of similar secure-email systems. In Section 5 we show the development of a formal theory in higher-order logic that describes the correctness of the email protocol: the theory states that the email protocol provides secure services to messages passing through it. We conclude in Section 6.

2 Overview of Cryptography

Network protocols rely on cryptographic algorithms to provide security services. Formal verification of these protocols requires formal definitions of not only the protocols themselves, but also the properties of the cryptographic algorithms they implore. Menezes and others have defined rigorously the terms related to cryptography functions such as the encryption scheme and the digital-signature scheme in [12]. Here we formalize cryptographic functions and their properties in HOL.

Before we get into any formula, we briefly describe how we have handled types.

2.1 Types and Type Conversion

There are many sets of entities exist in a cryptographic system, such as plaintexts, ciphertexts, keys and signatures. We view them as different types. A system can reject a value if it is not of a particular type. For example, if a system expects a key to be 128 bits long, then it will discard a value that is of 129 bits. We have modeled all the types in our work. When an entity is used for different purpose as different types, we use type converters which are constant functions to change types. For instance, a key is of type *key* when it is used to encrypted a message and it is of type *plaintext* when it is encrypted for transmission. We define a constant function *keyToPlaintext* to convert variables from type *key* to *plaintext*. If a variable *k* is of type *key*, then the type of *keyToPlaintext k* is *plaintext*.

For the simplicity of presentation we have ignored all types in this paper.

2.2 Encryption Scheme

Encryptions are used to protect the confidentiality of information. An encryption scheme consists of a set of encryption functions and a corresponding set of decryption functions. For each encryption function E, there is a unique decryption function D such that any message encrypted by E can be retrieved by D. We define *cipherPair* as a pair of uniquely associated encryption and decryption functions.

DEFINITION 1 (CIPHERPAIR) A pair of functions, E and D, is called a *cipherPair* if D is the unique left inverse of E.

$$\vdash_{def} \forall E\ D.\ cipherPair\ E\ D = (D \circ E = I) \wedge (\forall D\_arb.\ (D\_arb \circ E = I) \supset (D\_arb = D))$$

One way of designing an encryption scheme is to design one algorithm for the set of encryption functions and a corresponding algorithm for the set of decryption functions. Keys are used to pick out the particular encryption and decryption functions.

2.3 Digital-Signature Scheme

Signatures are used to identify principals. A digital-signature scheme consists of a set of signing functions and a corresponding set of signature verification functions. For any entity A, signing function S_A takes a message to a signature, while verification function V_A takes a message and a signature and returns a boolean value. Function S_A is kept secret by entity A, while V_A is made known to the public and is used by others to verify A's signatures.

For a pair of functions, S_A and V_A, to be consider secure, $V_A(m,s)$ should return *true* if and only if s is a valid signature of A on message m and if there is no practical way for any other entity to find a pair (m,s) such that $V_A(m,s)$ is *true*.

We define *DSPair* as a pair of uniquely associated signing and signature verification functions.

DEFINITION 2 (DSPAIR) A *DSpair* is a pair of functions—a signing function *Sign* and a verification function V—such that, for every message m, $V(m,s)$ is *true* if and only if s is a valid signature on m. The signing function *Sign* is a one-to-one function.

$$\vdash_{def} \forall Sign\ V.\ DSPair\ Sign\ V = (\forall m\ s.\ V\ (m,s) = (s = Sign\ m)) \wedge (\forall m1\ m2.\ (Sign\ m1 = Sign\ m2) \supset (m1 = m2))$$

A digital signature is uniquely associated with a signer and the information being signed, while a signature on paper is uniquely associated with a signer. When we move from paper signatures to digital signatures, we gain the ability to associate the information with a signature and, we lose the ability to uniquely identify a signer from the signature. With a digital signature, we can conclude that only entity A could have generated the signature on message m. However, it is not practical for anyone to fake a particular signature by a particular signer on chosen information.

Digital-signature schemes can be designed analogously for encryption schemes. One algorithm is designed for the set of signing functions, and a corresponding algorithm is designed for the set of verification functions. Keys are also used to pick out the particular signing and verification functions.

2.4 Secret-Key Cryptography

Secret-key cryptography uses the same key to specify its encryption and decryption transformations. We define *secKeyPair* to name the encryption function, decryption function, and the secret key used in *cipherPair*.

DEFINITION 3 (SECKEYPAIR) Functions *encryptS k* and *decryptS k* constitute a *cipherPair*.

$$\vdash_{def} \forall \mathit{encryptS}\ \mathit{decryptS}\ k.\ \mathit{secKeyPair}\ \mathit{encryptS}\ \mathit{decryptS}\ k = \mathit{cipherPair}\ (\mathit{encryptS}\ k)\ (\mathit{decryptS}\ k)$$

2.5 Public-Key Cryptography

Public-key cryptography uses two keys to specify its transformations: a private key, d_k, known only to the owner and a corresponding public key, e_k, accessible by the world. When used for an encryption scheme, the public key is used for encryption and the private key is used for decryption. When used for a signature scheme, the private key is used for signing and the public key is used for verification. These two keys form a unique key pair.

We define *pubKeyPair* to name the encryption function, the decryption function, and the pair of keys used in public key *cipherPair*.

DEFINITION 4 (PUBKEYPAIR) Functions *encryptS ek* and *decryptP dk* constitute a *cipherPair*.

$$\vdash_{def} \forall \mathit{encryptP}\ \mathit{decryptP}\ \mathit{ek}\ \mathit{dk}.\ \mathit{pubKeyPair}\ \mathit{encryptP}\ \mathit{decryptP}\ (\mathit{ek}, \mathit{dk}) = \mathit{cipherPair}\ (\mathit{encryptP}\ \mathit{ek})\ (\mathit{decryptP}\ \mathit{dk})$$

We define *DSKeyPair* to name the encryption function, the decryption function, and the pair of keys used in public key *DSPair*.

DEFINITION 5 (DSKEYPAIR) Functions *sign sk* and *verify vk* constitute a *DSPair*.

$$\vdash_{def} \forall \mathit{sign}\ \mathit{verify}\ \mathit{vk}\ \mathit{sk}.\ \mathit{DSKeyPair}\ \mathit{sign}\ \mathit{verify}\ (\mathit{vk},\mathit{sk}) = \mathit{DSPair}\ (\mathit{sign}\ \mathit{sk})\ (\mathit{verify}\ \mathit{vk})$$

3 Formal Definition of Security Services of Channels

A channel is a means of communication, a mechanism for entities to make statements [8]. A secure channel provides security services to messages such as confidentiality and source authentication, which are essential to network-system services such as establishing identities of entities and granting access to system resources.

To be able to formally analyze secure channels, we define the confidential channel and the source-authentic channel in this section.

A channel between a sender `A` and a receiver `B` consists of a sender process, a receiver process, and a network that transmits information from the sender process to the receiver process. Sender `A` makes a `statement` through a `package` generated by the sender process. Receiver `B` receives the `statement` recovered from the `package` by the receiver process(Figure 1). A package has the necessary header information for the particular services the channel provides.

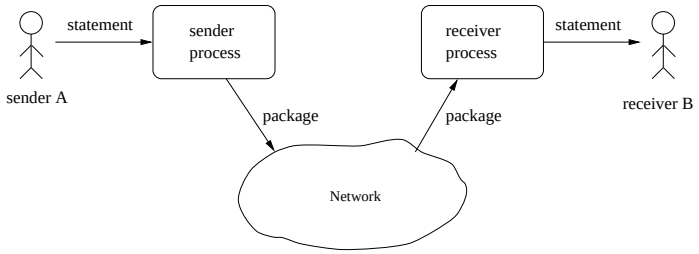

Fig. 1. A communication channel between entities `A` and `B`

3.1 Confidential Channel

A typical informal definition of confidentiality is as follows. Confidentiality implies that you know who the receiver is. A channel is confidential if the intended recipient can derive the statement from a received package while nobody else can. For example, sender `A` makes a statement *msgA* to receiver `B` through a communication channel consisting of the sender process *sendTo*, the receiver process *receiveByB*, and the network. Sender `A`'s process *sendTo* generates a package *envelopeA* and transmits it to `B`'s receiver process through the network. Entity `B`'s receiver process receives a package *envelopeB* and recovers a statement *msgB* using *receiveByB*. If the package *envelopeA* arrives intact at `B`'s process, then `B` recovers the statement *msgA*. Another entity `C`, which is also on the network, can observe to the package. However, even if the package *envelopeA* arrives intact at `C`'s process, `C` will not be able to recover the statement *msgA* (Figure 2).

Formal definition based on the above description is as follows.

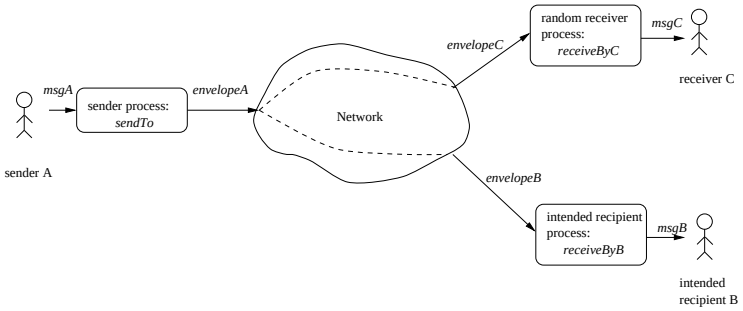

Fig. 2. A confidential channel

DEFINITION 6 (CONFCHANNEL) A confidential channel allows user A to send a statement through a package, knowing that regardless of who gets the package, only intended recipient B can read the statement in the package.

$$\vdash_{def} \forall sendTo\ receiveByB\ envelopeA\ msgA\ envelopeB\ msgB\ recipientB.$$
$$confChannel\ sendTo\ receiveByB\ envelopeA\ msgA\ envelopeB\ msgB\ recipientB =$$
$$(envelopeA = sendTo\ recipientB\ msgA) \supset$$
$$(((envelopeB = envelopeA) \supset$$
$$(msgB = receiveByB\ recipientB\ envelopeB) \supset$$
$$(msgB = msgA)) \wedge$$
$$(\forall receiveByC\ envelopeC\ msgC\ receiverC.$$
$$(envelopeC = envelopeA) \supset$$
$$(msgC = receiveByC\ receiverC\ envelopeC) \supset$$
$$(msgC = msgA) \supset$$
$$(receiverC = recipientB)))$$

3.2 Source Authentication Channel

Source authentication implies that you know who the real sender is. A channel adds source authentication to statements if the receiver process can derive the source of a received statement when the received package passes an authenticity check. For example, sender A makes a statement *msgA* to receiver B through a communication channel consisting of the sender process *sendFromA*, the network, the receiver process *receive*, and authenticity check *authChk*. Sender A sends a package *envelopeA* through *sendFromA* to B. Receiver B receives a package *envelopeB* and recovers a statement *msgB*. If the package *envelopeA* arrives intact at B's process, it will pass the authenticity check ($authChksenderA$) and B will recover the statement *msgA*. Suppose another entity D, which is also on the network and has full control of its process *sendFromD*, sends a package *envelopeD* to B and claims that it is from A. If the package *envelopeD* arrives

intact at B's process, it will not pass the authenticity check $(authChk\ senderA)$. This is illustrated in Figure 3. The formal definition of authentic channel is as follows.

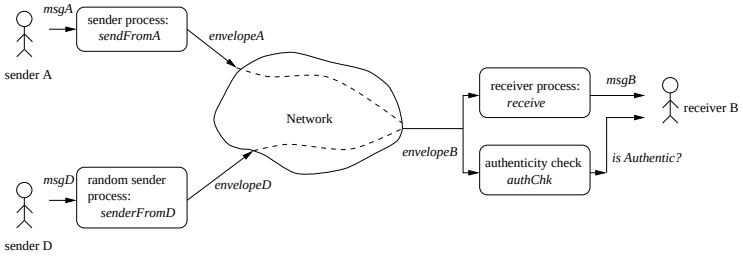

Fig. 3. A channel with source-authentication protection

DEFINITION 7 (AUTHCHANNEL) A channel provides source authentication service to a statement sent between A and B if it provides a way to certify the originator of the statement to the recipient.

$$
\begin{aligned}
\vdash_{def}\ & \forall authChk\ sendFromA\ receive\ envelopeA\ msgA\ envelopeB\ msgB\ senderA.\\
& authChannel\ authChk\ sendFromA\ receive\ envelopeA\ msgA\ envelopeB\\
& \quad msgB\ senderA =\\
& (msgB = receive\ envelopeB) \supset\\
& (((envelopeA = sendFromA\ senderA\ msgA) \supset\\
& \ (envelopeB = envelopeA) \supset\\
& \ (authChk\ senderA\ envelopeB \wedge (msgB = msgA))) \wedge\\
& \ (\forall sendFromD\ envelopeD\ msgD\ originatorD.\\
& \ (envelopeD = sendFromD\ originatorD\ msgD) \supset\\
& \ (envelopeB = envelopeD) \supset\\
& \ authChk\ senderA\ envelopeB \supset\\
& \ ((originatorD = senderA) \wedge (msgB = msgD))))
\end{aligned}
$$

A channel providing source-authentication service to statements also provides integrity service to the statements. If a statement in a package is corrupt, the source of the statement is the source of the corruption, hence the package will not pass the source authentication check.

4 Secure Core-Email Protocol

In the last section we studied the services of secure channels. In this and the next sections, we show one example channel—secure core-email protocol—that provides these

services. In this section we define the protocol rigorously. In the next section we verify that the protocol is secure.

Our example of secure channels is a secure core-email protocol. We have studied secure email systems PEM, PGP, and X.400. These systems differ from one another in message structures and the certificate hierarchies, among other things [7]. However, their cores that provide security services are the same. We extracted this core and named it "secure core-email protocol" (ScEP).

The secure core-email protocol provides confidentiality, message integrity, source authentication, and source non-repudiation services. It protects messages through a combination of secret-key encryption, public-key encryption, and digital-signature generation.

4.1 Sender Process

The sender process of ScEP is as follows. We refer to the content of an email as `message`. First, the process randomly generates a per-message data encryption key (DEK) and uses it as a secret key to encrypt the message. Second, it computes the message digest of the message using a hash function and computes the digital signature of the message by signing the message digest with the sender's private key. It then encrypts the digital signature with DEK. Last, the process encrypts DEK with the intended recipient's public key. The output of the sender process is a 4-tuple: (sender's public key, encrypted DEK, encrypted digital signature, encrypted message). Figure 4 illustrates the sender process.

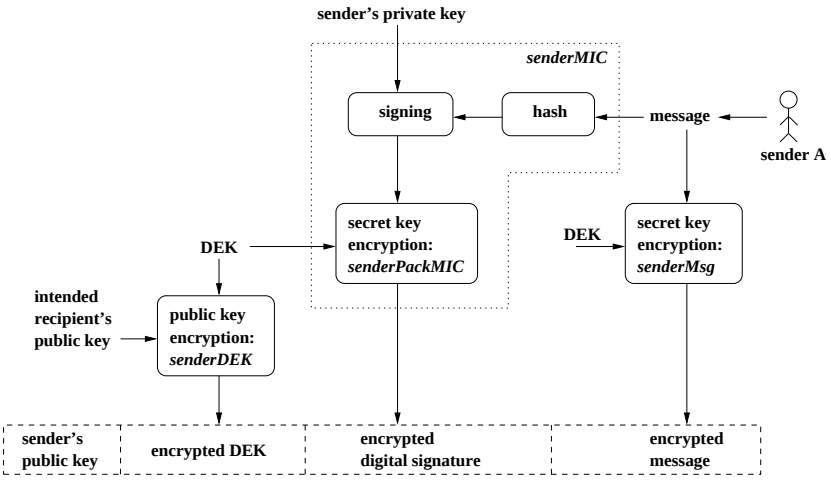

Fig. 4. Sender process of the secure core-email protocol: *enMailSender*

Table 1. Functions in the sender process *enMailSender*

Name	Definition and Description
senderDEK	packs DEK by encrypt it with receiver's public key: *senderDEK encryptP ekeyB DEK* = *encryptP ekeyB DEK*
senderMsg	packs a message by encrypt it with DEK: *senderMsg encryptS DEK message* = *encryptS DEK message*
senderGenMIC	generates a digital signature of a message by signing the message digest: *senderGenMIC sign hash skeyA message* = *sign skeyA* (*hash message*)
senderPackMIC	packs the digital signature of a message: *senderPackMIC encryptS DEK MIC* = *encryptS DEK MIC*
senderMIC	generates and packs the digital signature of a message: *senderMIC encryptS sign hash skeyA DEK message* = ((*senderPackMIC encryptS DEK*) *o* (*senderGenMIC sign hash skeyA*)) *message*

We define the sender process *enMailSender* in HOL as follows. The functions that appear in Figure 4 and in the definition of *enMailSender* are also defined in HOL and are listed in Table 1. The variables appeared in the definitions are described in Table 2.

DEFINITION 8 (ENMAILSENDER) Process *enMailSender* generates an email by encrypting and signing a message.

$\vdash_{def}$ $\forall$*encryptP encryptS sign hash vkeyA ekeyB DEK skeyA message.*
enMailSender encryptP encryptS sign hash vkeyA ekeyB DEK skeyA
message =
(*vkeyA*,
senderDEK encryptP ekeyB DEK,
senderMIC encryptS sign hash skeyA DEK message,
senderMsg encryptS DEK message)

4.2 Receiver Process

The receiver process of ScEP reverses the sender process to recover the message. The receiver process expects a 4-tuple as input, the same 4-tuple that is the output of the sender process. To process a received email, the receiver process first accesses the fields of the email to get the sender's public key, encrypted DEK, encrypted digital signature, and the encrypted message. The receiver process then recovers the per-message encryption key DEK by decrypting the encrypted DEK using the receiver's private key. It then uses DEK to retrieve message and digital signature by decrypting the encrypted message and the encrypted digital signature respectively. Finally, it checks the trustworthiness of the received message by checking the recovered digital signature against

Table 2. Variables in the sender and receiver processes

Name	Description
DEK	data encryption key
decryptP	public key decryption function
decryptS	secret key decryption function
ekeyB	receiver's public key (for encryption)
enDEK	encrypted DEK
enMIC	encrypted digital signature
enMsg	encrypted message
encryptP	public key encryption function
encryptS	secret key encryption function
envelope	email, 4-tuple with sender's public key
flag	indication of the trustworthiness of a received email
hash	hash function
message	content of an email
privateKey	constant function, naming the corresponding private key given a public key
rxEnvelope	received email
rxMessage	received message in *rxEnvelope*
skeyA	sender's private key (for message signing)
sign	signing function
txEnvelope	transmitted email
txMessage	transmitted message in *txEnvelope*
verify	signature verification function
vkeyA	sender's public key (for signature verification)

the recovered message: it computes the message digest of the message using the hash function and verifies the digital signature against the message digest using the sender's public key. The receiver process is illustrated in Figure 5.

We define the receiver process *enMailReceiver* in HOL as follows. The functions that appear in Figure 5 and in the definition of *enMailReceiver* are also defined in HOL and are listed in Table 3. The variables appeared in the definitions are described in Table 2.

DEFINITION 9 (ENMAILRECEIVER) Process *enMailReceiver* retrieves *message* from an encrypted-signed mail.

$$\vdash_{def} \forall \textit{decryptP decryptS verify hash dkeyB envelope.}$$
$$\textit{enMailReceiver decryptP decryptS verify hash dkeyB envelope} =$$
$$(\textit{enMailVerMIC decryptP decryptS verify hash dkeyB envelope},$$
$$\textit{enMailRetMsg decryptP decryptS dkeyB envelope})$$

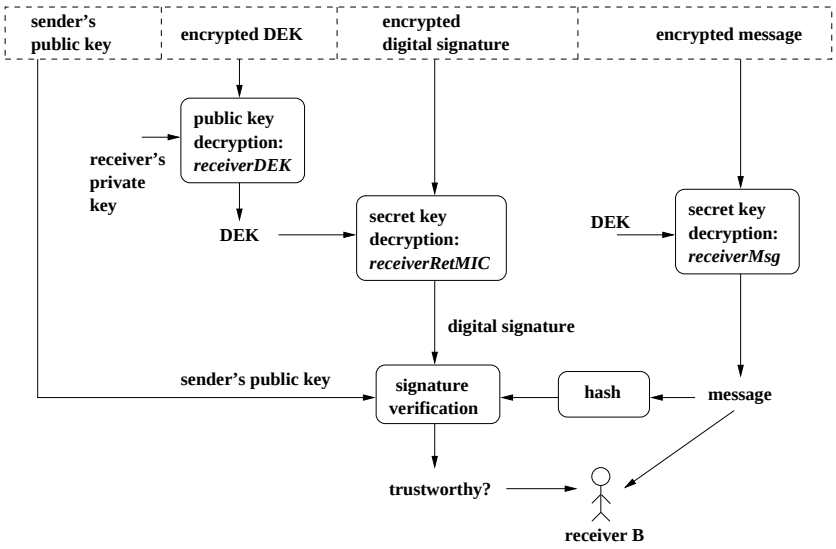

Fig. 5. Receiver process of secure core-email protocol: *enMailReceiver*

4.3 Secure Email System

A system that follows ScEP consists of a sender process and a receiver process. The cryptographic functions used by the sender and receiver processes must have proper properties. The required properties of these functions are:

- $(encryptSDEK), (decryptSDEK)$ comprises a $cipherPair$. DEK is a secret key.
- $(encryptPekeyB), (decryptPdkeyB)$ comprises a $cipherPair$. The pair $(ekeyB, dkeyB)$ is a public-key pair.
- $(signskeyA), (verifyvkeyA)$ comprises a $DSpair$. The pair $(vkeyA, skeyA)$ is a public-key pair.
- The received email contains a valid public key for signature verification, and the public key has a corresponding private key for signing messages.

We define secure email system in HOL as follows. The function *privateKey* names the corresponding private key given a public key. It is defined as a constant in HOL. The variables appeared in the definitions are described in Table 2.

DEFINITION 10 (ENMAILSYSTEM) Encrypted-signed-message mail system *enMailSystem* consists of the sender process *enMailSender* and the receiver process *enMailReceiver*, whose keys make up digital signature key pairs and cipher key pairs.

$\vdash_{def}$ $\forall$*encryptP encryptS decryptP decryptS sign verify hash vkeyA skeyA ekeyB dkeyB DEK txEnvelope txMessage rxEnvelope rxMessage flag*.

Table 3. Functions in the receiver process *enMailReceiver*

Name	Definition and Description
receiverDEK	retrieves DEK from encrypted version: *receiverDEK decryptP dkeyB enDEK* = *decryptP dkeyB enDEK*
receiverMsg	retrieves message from encrypted version: *receiverMsg decryptS DEK enMsg* = *decryptS DEK enMsg*
receiverRetMIC	retrieves digital signature from encrypted version: *receiverRetMIC decryptS DEK enMIC* = *decryptS DEK enMIC*
receiverVerMIC	verifies digital signature against the message: *receiverVerMIC verify hash vkeyA message MIC* = *verify vkeyA* (*hash message, MIC*)
receiverMIC	retrieves digital signature and verifies it: *receiverMIC decryptS verify hash vkeyA DEK message enMIC* = ((*receiverVerMIC verify hash vkeyA message*) *o* (*receiverRetMIC decryptS DEK*)) *enMIC*
enMailVerMIC	verifies the trustworthiness of a received mail: *enMailVerMIC decryptP decryptS verify hash dkeyB envelope* = (*let* (*vkeyA, enDEK, enMIC, enMsg*) = *envelope in* *let DEK* = *receiverDEK decryptP dkeyB enDEK in* *let message* = *receiverMsg decryptS DEK enMsg in* *receiverMIC decryptS verify hash vkeyA DEK message enMIC*)
enMailRetMsg	retrieves the message from a received mail: *enMailRetMsg decryptP decryptS dkeyB envelope* = (*let* (*vkeyA, enDEK, enMIC, enMsg*) = *envelope in* *let DEK* = *receiverDEK decryptP dkeyB enDEK in* *receiverMsg decryptS DEK enMsg*)
enMailRetSender	retrieves the sender's public key from a received mail: *enMailRetSender envelope* = (*let* (*vkeyA, enDEK, enMIC, enMsg*) = *envelope in vkeyA*)

enMailSystem encryptP encryptS decryptP decryptS sign verify
hash vkeyA skeyA ekeyB dkeyB DEK txEnvelope txMessage
rxEnvelope rxMessage flag =
(*txEnvelope* =
enMailSender encryptP encryptS sign hash vkeyA ekeyB DEK
skeyA txMessage) ∧
((*flag,rxMessage*) =
enMailReceiver decryptP decryptS verify hash dkeyB rxEnvelope) ∧
secKeyPair encryptS decryptS DEK ∧
pubKeyPair encryptP decryptP (*ekeyB, dkeyB*) ∧
DSKeyPair sign verify (*vkeyA, skeyA*) ∧
DSKeyPair sign verify

(*enMailRetSender rxEnvelope*,
privateKey (*enMailRetSender rxEnvelope*))

To simplify the protocol we have ignored the selection of cryptographic functions used by the sender and receiver processes. However, in the HOL definitions of these two processes the cryptographic functions are taken as parameters.

We have also ignored the necessary verification of certificates. A certificate is a document issued by a certificate authority certifying an entity's public key, much like the entries in telephone directory. A certificate contains an entity's name and public key and is signed by the certification authority. Anyone with certificate authority's public key can verify a certificate, hence can establish a channel where a public key speaks for the entity. In ScEP we identify as the source of an email the public key of an entity, not the entities itself.

In ScEP, the sender's public-key pair is used for signing and signature verification; the receiver's public-key pair is used for encryption and decryption. It is possible that entities in the network use different public-key pairs for different purposes: one pair for signing and signature verification, and one pair for encryption and decryption.

5 Formal Verification of Secure Communication Channels

In the last section we formally defined the ScEP system. A ScEP system can be regarded as a channel between a sender and a receiver that provides confidentiality and source authentications to the statements. In ScEP, a sender identifies an intended recipient of a statement with the recipient's public key and, a receiver identifies the source of a statement with the sender's public key contained in the received package. The channel between a sender `A` and a receiver `B` is broken down into three sub-channels: a channel C_A between `A` and a key k_A that `A` holds, a channel C_B between `B` and a key k_B that `B` holds, and a channel $C_{k_A k_B}$ between k_A and k_B. The composition of these three channels is channel C_{AB} between entity `A` and `B`. (Figure 6.)

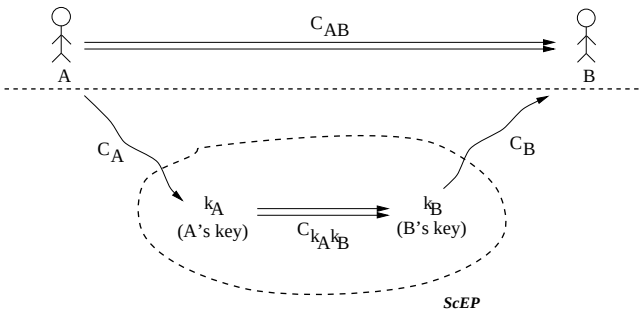

Fig. 6. Communication channels between entities `A` and `B`

In this work we concentrate on the analysis of ScEP, which is a channel between two entities' keys. In Section 3 we defined confidential and source authentic channels based on entities. To verify that this channel provides secure services, we redefine the confidential and source authentic channels to be based on keys.

5.1 Confidential Channel

Because both public-key and secret-key encryption are used in ScEP and they use keys differently, we redefine confidential channels for each case. With a public-key encryption a channel is confidential if, when `A` sends `B` a statement encrypted with `B`'s public key, only the entity with `B`'s private key can recover the statement in the package. With a secret-key encryption a channel is confidential if, when `A` sends `B` a statement encrypted with a secret key *k*, only the entity knows the secret *k* can recover the statement in the package. The definitions of these two confidential channels in HOL are as follows.

DEFINITION 11 (CONFCHANNELPUB) A channel is confidential if only the entity with knowledge of the intended recipient's private key can read the statement in the package. Parameters *ekeyB* and *dkeyB* respectively denote the public and private keys of the intended recipient. Parameter *keyC* denotes a quantity that an arbitrary entity `C` uses to retrieves the statement.

$$
\begin{aligned}
\vdash_{def}\ & \forall \mathit{sendTo}\ \mathit{receiveByB}\ \mathit{envelopeA}\ \mathit{msgA}\ \mathit{envelopeB}\ \mathit{msgB}\ (\mathit{ekeyB}, \mathit{dkeyB}).\\
& \mathit{confChannelPUB}\ \mathit{sendTo}\ \mathit{receiveByB}\ \mathit{envelopeA}\ \mathit{msgA}\\
& \mathit{envelopeB}\ \mathit{msgB}\ (\mathit{ekeyB}, \mathit{dkeyB}) =\\
& (\mathit{envelopeA} = \mathit{sendTo}\ \mathit{ekeyB}\ \mathit{msgA}) \supset\\
& (((\mathit{envelopeB} = \mathit{envelopeA}) \supset\\
& (\mathit{msgB} = \mathit{receiveByB}\ \mathit{dkeyB}\ \mathit{envelopeB}) \supset\\
& (\mathit{msgB} = \mathit{msgA})) \wedge\\
& (\forall \mathit{receiveByC}\ \mathit{envelopeC}\ \mathit{msgC}\ \mathit{keyC}.\\
& (\mathit{envelopeC} = \mathit{envelopeA}) \supset\\
& (\mathit{msgC} = \mathit{receiveByC}\ \mathit{keyC}\ \mathit{envelopeC}) \supset\\
& (\mathit{msgC} = \mathit{msgA}) \supset\\
& (\mathit{keyC} = \mathit{dkeyB})))
\end{aligned}
$$

DEFINITION 12 (CONFCHANNELSEC) A channel is confidential if only the entity with knowledge of a shared secret key can read the statement in the package. Parameter *secretAB* is the shared secret between the sender and the intended recipient. Parameter *keyC* denotes a quantity that an arbitrary entity `C` uses to retrieve the statement.

$$
\begin{aligned}
\vdash_{def}\ & \forall \mathit{sendTo}\ \mathit{receiveByB}\ \mathit{envelopeA}\ \mathit{msgA}\ \mathit{envelopeB}\ \mathit{msgB}\ \mathit{secretAB}.\\
& \mathit{confChannelSec}\ \mathit{sendTo}\ \mathit{receiveByB}\ \mathit{envelopeA}\ \mathit{msgA}\ \mathit{envelopeB}\ \mathit{msgB}\ \mathit{secretAB} =\\
& (\mathit{envelopeA} = \mathit{sendTo}\ \mathit{secretAB}\ \mathit{msgA}) \supset\\
& (((\mathit{envelopeB} = \mathit{envelopeA}) \supset\\
& (\mathit{msgB} = \mathit{receiveByB}\ \mathit{secretAB}\ \mathit{envelopeB}) \supset\\
& (\mathit{msgB} = \mathit{msgA})) \wedge
\end{aligned}
$$

$(\forall receiveByC\ envelopeC\ msgC\ keyC.$
$(envelopeC = envelopeA) \supset$
$(msgC = receiveByC\ keyC\ envelopeC) \supset$
$(msgC = msgA) \supset$
$(keyC = secretAB)))$

However, with Definitions 11 and 12 of confidential channel, we are unable to prove the ScEP provides the confidentiality services. There are two reasons:

1. There are several ways to identify entities [6]. The previous definitions used "what an entity knows" (e.g. *dkeyB*) to identify the entity. This is not suitable for our model. A better alternative is to use "what an entity can do" (e.g. *receiveByB dkeyB*) to identify the entity. The definition of $confChannelPub$ is rewritten as an example:

   DEFINITION 13 (CONFCHANNELPUB') Definition of confidential channel where an entity is identified by "what he can do".

   $\vdash_{def}\ \forall sendTo\ receiveByB\ envelopeA\ msgA\ envelopeB\ msgB\ (ekeyB, dkeyB).$
   $confChannelPub'\ sendTo\ receiveByB\ envelopeA\ msgA\ envelopeB\ msgB$
   $(ekeyB,dkeyB) =$
   $\forall receiveByC\ envelopeC\ msgC\ keyC.$
   $(envelopeA = sendTo\ ekeyB\ msgA) \supset$
   $(envelopeC = envelopeA) \supset$
   $(msgC = receiveByC\ keyC\ envelopeC) \supset$
   $((msgC = msgA) = (receiveByC\ keyC = receiveByB\ dkeyB))$

2. To prove the ScEP is a confidential channel, we need to assume that the encryption and decryption functions used satisfy the following property:

   DEFINITION 14 (CIPHERPROP) If E and D constitute a *cipherPair*, then D is the only function that can decipher a message encrypted by E.

   $\vdash_{def}\ \forall E\ D.$
   $cipherProp\ E\ D =$
   $cipherPair\ E\ D \supset$
   $(\forall m\ D\_arb.\ (D\_arb\ (E\ m) = m) \supset\ (D\_arb = D))$

   However, any constant function D_arb is going to satisfy $(D\_arb\ (E\ m) = m)$ for some value m. Therefore, there is no pair of functions E and D that has the property $cipherProp$.

5.2 Source Authentication Channel

We redefine a source-authentic channel based on public-key cryptography because only the public-key digital signature is used in ScEP. A channel provides source-authentic service to a statement if, only when A sends B a statement sealed with A's signature will the channel certifies the statement as coming from A. In Definition 7 we derive the source of a statement according to how a package is generated from the statement. One way of generating a package, usually adopted by a person of authority, is to check the validity of a statement and signs it to indicate the source of the statement. We refine the source-authentic channel based on this approach in HOL:

DEFINITION 15 (AUTHCHANNELDS) A channel provides source authentication services to statements if it certifies the origins of the received statements. Parameter *sealA* denotes sender A's action of validating a statement and signing it. Parameter *sealD* denotes a function that an arbitrary entity D uses to sign a statement. The function *retSeal* retrieves the seal of the mail and the function *retSender* retrieves the public key of the sender. Function *authChk* is the authentication check of the mail.

$$
\begin{aligned}
\vdash_{def}\ & \forall authChk\ sealA\ retSeal\ retSender\ envelopeA\ msgA\ envelopeB\ msgB \\
& \quad vkeyA\ skeyA. \\
& \quad authChannelDS\ authChk\ sealD\ retSeal\ retSender\ envelopeA\ msgA \\
& \qquad envelopeB\ msgB\ (vkeyA,skeyA) = \\
& \quad (\forall sealD\ envelopeD\ msgD\ keyD. \\
& \qquad (envelopeB = envelopeD) \supset \\
& \qquad (retSeal\ envelopeD = sealD\ keyD\ msgD) \supset \\
& \qquad (vkeyA = retSender\ envelopeB) \supset \\
& \qquad (authChk\ envelopeB = \\
& \qquad\ (sealD\ keyD\ msgD = sealA\ skeyA\ msgB)))
\end{aligned}
$$

In this definition we equate two entities by their ability to generate a particular signature s such that $verifyvkeyA(msgB, s)$ is $true$. A stronger equivalence between two entities would be by equating their signing ability such as *signAskeyA* and *signDkeyD*. This is necessary for the following theorem because, as discussed in Section 2.3, a digital signature is uniquely associated with a signer and the information being signed, rather than with a signer alone.

The following theorem shows that a ScEP system provides a source-authentic channel to statements.

THEOREM 16 (ENAUTHENTIC) A ScEP system provides a source authentication channel to statements.

$$
\begin{aligned}
\vdash\ & enMailSystem\ encryptP\ encryptS\ decryptP\ decryptS\ sign\ verify\ hash \\
& \quad vkeyA\ skeyA\ ekeyB\ dkeyB\ txDEK\ envelopeA\ msgA \\
& \quad envelopeB\ msgB\ flag \supset \\
& \quad authChannelDS\ (enMailVerMIC\ decryptP\ decryptS\ verify\ hash\ dkeyB) \\
& \qquad (senderGenMIC\ sign\ hash) \\
& \qquad (enMailRetMIC\ decryptP\ decryptS\ dkeyB) \\
& \qquad enMailRetSender\ envelopeA\ msgA\ envelopeB\ msgB\ (vkeyA,skeyA) \wedge \\
& \quad (flag = \\
& \quad enMailVerMIC\ decryptP\ decryptS\ verify\ hash\ dkeyB\ envelopeB)
\end{aligned}
$$

6 Conclusion

Our objectives were to specify security properties and their implementation mechanisms, so we could prove the implementation mechanisms satisfied the desired properties. These mechanisms form the core of several secure email protocols such as PGP and PEM. The services we looked at were confidentiality and source authenticity. At

this time we have proved the implementations satisfy the source-authenticity service. We are currently working on verifying confidentiality service. This will likely require a reformulation of confidentiality as it relates to implementation.

References

1. M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, January 1999. Also appeared as SRC Research Report 149.
2. Stephen H. Brackin. A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols. In *Proceedings of 9th IEEE Computer Security Foundation Workshop*, pages 62–76, June 1996.
3. M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990. Also appeared as SRC Research Report 39.
4. M.J.C. Gordon. Why Higher-Order Logic is a Good Formalism for Specifying and Verifying Hardware. In G. Milne and P.A. Subrahmanyam, editors, *VLSI specification, verification and synthesis*. North Holland, 1986.
5. 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.
6. Roberto Gorrieri and Paul Syverson. Varieties of Authentication. In *Proceedings of 11th IEEE Computer Security Foundations Workshop*, pages 79–82, 1998.
7. Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security Private Communication in a Public World*. Prentice Hall, New Jersey, 1995.
8. Butler Lampson, Martin Abadi, Michael Burrows, and Edward P. Wobber. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992. Also appeared as SRC Research Report 83.
9. J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC 1421, DEC, February 1993. ftp: ds.internic.net.
10. G. Lowe. Casper: A Compiler for the Analysis of Security Protocols. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 18–30, 1997.
11. Catherine A. Meadows. The NRL Protocol Analyzer: An Overview. In *The Proceedings of Second International Conference on the Practical Applications of Prolog*, April 1994.
12. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, New York, 1996.
13. R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21:993–999, 1978.
14. Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the USENIX Winter Conference*, pages 191–202, February 1988.
15. Dan Zhou, Joncheng C. Kuo, Susan Older, and Shiu-Kai Chin. Formal Development of Secure Email. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, January 1999.
16. P.R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, Massachusetts, 1995.