Certified Mailing Lists

Himanshu Khurana
National Center for Supercomputing Applications
University of Illinois
Urbana-Champaign, IL, USA
hkhurana@ncsa.uiuc.edu

Hyung-Seok Hahm
Electrical and Computer Engineering Dept.
University of Illinois
Urbana-Champaign, IL, USA
hahm1@uiuc.edu

ABSTRACT

Email List Services (or simply, *mailing lists*) are becoming increasingly common for collaborative computing. In order to enable their use for official purposes with increased effectiveness and security, services typically provided by postal mail (e.g. fair delivery) need to be provided in mailing lists. In this paper we propose a novel Certified Mailing-list Protocol (CMLP) that provides fair delivery, confidentiality, non-repudiation of origin and receipt, and authentication and integrity. We have formally specified and verified the CMLP protocol with Proverif, which is a fully-automated protocol verification tool.

Categories and Subject Descriptors

H.4.3 [**Communications Applications**]: Electronic Mail

Keywords

Mailing lists, Certified delivery

1. INTRODUCTION

As more and more user communities are engaging in collaborative tasks, use of Email List Services (or simply *Mailing Lists* - MLs) is becoming common; i.e., emails exchanged with the help of a list server (an example of a commonly used list server software is Majordomo). In addition to being a commonly used means of establishing collaborative groups, MLs are also an integral component of many groupware applications such as document annotation and storage [26] [9], mobile teamwork [22], and software development [19]. The increasing popularity of mailing lists (MLs) for exchanging both public and private information content can be gauged from the fact that there are over 300,000 registered LISTSERV lists while only 20% of those serve public content [13]. In addition to supporting exchange of private information content, MLs can be used for dissemination and exchange of official documents (e.g., in multi-party contract negotiations and signing, official announcements). However, in order to effectively support such dissemination and exchange, services similar to those provided by postal mail such as fair delivery, non-repudiation, and confidentiality are needed. In other words, there is a need to develop *certified mailing lists*.

Certified email for two-party email exchange is a well studied problem and aims to ensure that the recipient gets the email content if and only if the sender gets a proof-of-receipt from the recipient. In addition, this exchange is often confidential in that only the sender and receiver can see the email content. Certified email protocols derive from a larger family of protocols for fair exchange of digital goods and are close in nature to fair non-repudiation protocols, which are an instance of fair exchange. Certified email protocols fall into two categories: *inline* or *optimistic*. Inline protocols employ a Trusted Third Party (TTP) that is actively involved in message exchange while in optimistic protocols the sender and receiver exchange messages directly and rely on the TTP only for dispute resolution. Many inline (e.g., [14], [28], [16], [3]) as well as optimistic (e.g., [4], [7], [11]) protocols for two-party exchange have been proposed. Recently, several multi-party fair exchange, fair non-repudiation, and certified email protocols have been proposed as well. Of these ([17], [23]) are inline while ([5], [8], [24], [15], [27]) are optimistic.

MLs are popular largely because they offload the management of list subscriber information (e.g. email addresses) from users to the list server and significantly reduce user overheads. Certified MLs must retain this ease in that subscribers should be able to offload the management of necessary keying material and message processing that provide certified email delivery. This unique challenge is not satisfied by existing multi-party certified email optimistic ([15], [27]) and inline solutions [23] all of which require the sender to process individual receipts from the recipients and maintain public keys of every recipient to encrypt emails for confidentiality.

In this paper we develop a novel inline certified mailing list protocol that retains the ease of use of existing ML systems. Inline protocols are more suitable than optimistic protocols for two reasons. First, MLs already use an on-line entity (i.e. the list server) for email exchange. Second, only an on-line entity can be used to offload message processing for certified delivery such that subscriber overhead is minimized. In fact, our protocol uses a TTP that is co-located with the list server for simplicity (but this is not necessary for the correct operation of the protocol). The protocol provides the necessary assurances for certified delivery while minimizing subscriber overhead. We also formally verify the correctness of our protocol using Proverif [2], which is a fully-automated protocol verification tool.

The rest of this paper is organized as followed. In section 2 we detail the requirements of a certified mailing list protocol. In Section 3 we present our protocol and analyze it informally. In Section 4 we discuss the overhead imposed by the the protocol and give examples of its use. In Section 5 we formally verify the protocol using Proverif and we conclude in Section 6.

2. REQUIREMENTS AND APPROACH

Certified email protocols are an instance of fair exchange protocols where the recipient obtains an email if and only if the sender obtains a receipt. The aim of such protocols is to be resistant to possible attempts at cheating by the involved parties. Furthermore, when a dispute arises, all parties should be able to provide irrefutable evidence towards the resolution of the dispute. In this section we first discuss the entities that would be involved in the Certified Mailing List Protocol (CMLP). We then define the requirements of certified mailing list protocols. Blundo *et al.* [11] provide a more comprehensive list of requirements of certified email protocols dealing with two party email exchange and while we do not include all those requirements, we do include ones that distinguish mailing lists from two party email exchange. We then discuss our approach in satisfying these requirements.

The following entities are involved in the CMLP:

- *List Moderator (LM).* LM is a user (or process) that creates a list to be maintained at the list server, authenticates users, and helps them subscribe to and unsubscribe from the list. To create a list, LM establishes key material with the list server, and to subscribe a user, it distributes key material to the joining user. LM signs and distributes certificates of list members, and provides certificate validation with respect to list membership. We assume that LM is an autonomous entity; in particular, it is not controlled by the list server.
- *List Server (LS).* LS creates lists, maintains membership information (e-mail addresses and key material), adds and removes subscribers based on information received from LM, and forwards e-mails sent by a valid list subscriber to all current subscribers of that list.
- *Trusted Third Party (TTP).* TTP is directly involved in exchange of messages between sender and receivers. It is responsible for ensuring fairness by providing recipients with email contents and senders with receipts. For simplicity we co-locate the TTP with LS and call the entity L/T; however, this co-location is not necessary for the correct operation of our protocol as we discuss later.
- *Users/Subscribers.* Users subscribe to lists by sending join requests to LM, obtaining key material from LM, and then sending key material to L/T to complete the subscription process. Users send and receive certified email with the help of L/T.

The following are the requirements and our approach in addressing them for the CMLP:

Strong Fairness. Strongly fair certified mailing list protocols must not allow any party to interrupt or corrupt a protocol in an effort to gain any advantage from the exchanged messages. At the completion of a *strongly fair* certified mailing list protocol all parties should get the information they desired or no one should be able to obtain any useful information. This implies that a receiver (i.e. a list subscriber) gets access to an email message if and only if *all* list subscribers get the message *and* the sender obtains a receipt from *every* subscriber of that list.

Approach. The basic design choice of inline protocol allows us to use L/T for providing strong fairness in CMLP. L/T provides strong fairness by forwarding the key that can be used to decrypt a given email to all list subscribers only when it is able to give the email sender a receipt from all subscribers for that message.

Weak Fairness. Weakly fair certified mailing list protocols must not allow any party to interrupt or corrupt a protocol in an effort to gain any advantage from the exchanged messages. At the completion of a *weakly fair* certified mailing list protocol every pair of sending and receiving parties should get the information they desired or no one should be able to obtain any useful information. This implies that a receiver (i.e. a list subscriber) gets access to an email message if and only if the sender obtains a receipt from at least that receiver. This weaker notion of fairness may suffice for some applications (e.g. information propagation) but not for others (e.g. multi-party contract negotiations and signing)[1].

Approach. Again, we use L/T to provide weak fairness in CMLP. L/T does so by forwarding the key that can be used to decrypt an email message to a given list subscriber only when it is able to give the email sender a receipt from that subscriber for that message.

Confidentiality. Only the sender and the receivers should be able to extract the plaintext contents of email messages. This implies that the list server and any other entity involved in the protocol should not be able to access the message contents.

Approach. This requirement deserves a closer look as it has been addressed in several different ways to varying extents in the literature. In two-party certified email protocols it has been deemed important that the TTP (whether inline or optimistic) should not be able to read emails as part of the confidentiality requirement. To address this goal Blundo *et al.* [11] require the sender to encrypt the email with the receiver's public key while in order to reduce the PKI overhead, Abadi *et al.* [3] assume a partially trusted TTP and use the TTP's public key for encrypting emails. In multi-party certified email confidentiality of emails has been used as a means to achieve fairness in that the key encrypting the email is made available to the recipients only when the sender obtains receipts from the recipients. To that end, Zhou [27] requires the sender to encrypt the symmetric email encryption key individually with each recipient's public key while Kremer and Markowitch [23] use a group encryption scheme that allow the sender to compute a single encryption for all recipients.

When using mailing lists for multi-party certified email we need to be concerned about two entities outside the set of sender and recipients, namely, LS and TTP. Our goal is to ensure that neither LS nor TTP can read the emails. However, preventing the list server that forwards emails to the list subscribers from being able to read those emails is a

[1]Note that this notion of strong and weak fairness is different from that for non-repudiation protocols defined in [23].

tricky problem. Khurana *et al.*[21] show in their SELS protocol that we can solve this problem and address the confidentiality requirement via a novel software-based proxy encryption scheme presented in the paper. This proxy encryption scheme allows L/T to transform messages between the sender and receivers without requiring access to the email plaintext or to private keys that can be used to decrypt the email messages. Furthermore, list subscribers only have to maintain one public encryption/decryption key pair. Other approaches such as those undertaken by multi-party certified email protocols would require list subscribers to maintain public encryption/decryption key pairs for all recipients, which goes against the mailing list approach. To address confidentiality in CMLP, we use proxy encryption and present the proxy encryption scheme for CMLP in Section 3.

Non-repudiation of origin. Any party that originates a given message should not be able to falsely deny having originated it.

Approach. We use digital signatures to provide non-repudiation of origin. The sender must sign outgoing emails and receivers can provide emails along with the sender's signature to ensure non-repudiation of origin.

Non-repudiation of receipt. The recipient of a message should not be able to falsely repudiate the fact that she received the message.

Approach. We use digital signatures to provide non-repudiation of receipt. The receiver must sign email receipts and the sender can provide the email along with the receiver's signature on the receipt to ensure non-repudiation of receipt.

Authenticity and Integrity. Parties involved in the protocol should be able to verify each other's identities and should not be able modify messages without such modifications being detected.

Approach. We use digital signatures to provide authentication and integrity. All protocol messages are signed by the entity sending the message and the recipients of these messages can verify the digital signature to authenticate the sender and to ensure that the messages have not been modified in transit.

3. PROTOCOL

In the section, we describe our inline Certified Mailing-list Protocol (CMLP). In keeping with the requirements and approach outlined in the previous section the CMPL protocol combines properties of multi-party inline fair exchange and fair non-repudiation protocols with SELS [21]. Earlier work in multi-party inline fair exchange [17] introduces the notion of a *semi trusted* inline third party where the third party is not trusted with access to the (digital) goods being exchanged (it is, however, trusted to execute the fair exchange protocol correctly). More recently work in inline fair non-repudiation [23], which is closer in nature to certified email than fair exchange of digital goods, presents an approach that is suitable for CMLP. That is, to encrypt the email with a symmetric key and distribute the key to only those recipients that have provided the sender with a receipt. Furthermore, this work retains the notion of a semi trusted third party even though that is not an explicitly stated goal. To achieve fair non-repudiation the protocol requires the sender to send two messages − one to the receivers directly and one to the TTP. We believe that this can be improved upon in the mailing list setting and we achieve fair exchange of certified email in a more optimized way with the sender only needing to send one message to L/T (this is possible because we combine fair exchange of certified email with the proxy encryption scheme of SELS).

Instead of designing layered protocols where a fair exchange protocol would be placed on top of SELS, we develop an integrated protocol for two reasons. First, we provide a modification to the proxy encryption scheme that allows L/T to deliver messages with one less exponentiation than that required by SELS (but does not affect the security of the scheme). This modification also affects the list subscription component of the SELS protocol. Second, we feel that an integrated protocol provides for an easier explanation of certified mailing lists. In the following description of the CMLP protocol the list creation and subscription components are similar to those described in SELS while the component for sending certified email is very different from the one that involves sending of emails in SELS as it provides fair exchange of certified email between the sender and the other list subscribers.

3.1 CMLP Proxy Encryption Scheme

The CMLP public-key encryption scheme $\mathcal{E}$ is based on the discrete log problem like El Gamal. $\mathcal{E}$ specifies an encryption function that enables L/T to transform an e-mail message encrypted with the sender's public-key into messages encrypted with the receivers' public keys. We first present a notation for El Gamal and then describe $\mathcal{E}$.

Let $\mathcal{E}_{eg} = (Gen, Enc, Dec)$ be the notation for standard El Gamal encryption. Gen is the key generating function. Hence $Gen(1^k)$ outputs parameters (g, p, q, a, g^a) where g, p and q^2 are group parameters, (p being k bits), a is the private key, and $y = g^a \bmod p$ is the public key. The Enc algorithm is the standard El Gamal encryption algorithm and is defined as $e = (mg^{ar} \bmod p,\ g^r \bmod p)$, where r is chosen at random from Z_q. To denote the action of encrypting message m with public key y, we write $Enc_{PK_y}(m)$. Dec is the standard El Gamal decryption algorithm and requires dividing mg^{ar} (obtained from e) by $(g^r)^a \bmod p$. We assume all arithmetic to be *modulo p* unless stated otherwise.

The CMLP asymmetric encryption scheme is denoted by $\mathcal{E} = (IGen,\ UGen, AEnc, ADec, \Gamma)$. With further details in Section 3.2, $IGen$ is a distributed protocol executed by LM and L/T to generate group parameters g, p and q, private keys K_{LM} and $K_{L/T}$ and public keys $PK_{LM} = g^{K_{LM}}$, $PK_{L/T} = g^{K_{L/T}}$, and $PK_{LK} = g^{K_{LM}} g^{K_{L/T}}$. K_{LM} is simply a random number in Z_q chosen by LM, and $K_{L/T}$ is a random number chosen by L/T. $UGen$ is a distributed protocol executed by user U_i, LM, and L/T to generate private keys for U_i and proxy keys for L/T. $UGen(K_{LM}, K_{L/T})$ outputs private key K_{U_i}, proxy key K'_{U_i} and the public keys $PK_{U_i} = g^{K_{U_i}}$, $PK'_{U_i} = g^{K'_{U_i}}$. K'_{U_i} is called user U_i's *proxy key* and is held by L/T. Furthermore, it is guaranteed that $K_{U_i} + K'_{U_i} = K_{LM} + K_{L/T} \bmod q$. This protocol requires U_i, LM, and L/T to generate random numbers and add/subtract them from K_{LM} and $K_{L/T}$. $AEnc$ and $ADec$ are identical to Enc and Dec defined above. $\Gamma_{K'_{U_i}}$ is a transformation function that uses user U_i's proxy key to transform messages encrypted with PK_{LK} into messages encrypted with

^{2}Using subgroups ensures that El Gamal is semantically secure [12].

user U_i's public key. It takes as input an encrypted message of the form $(g^{rK_{LK}}M, g^r)$ and outputs $(g^{rK_{U_i}}M, g^r) = ((g^{rK'_{U_i}})^{(-1)} g^{rK_{LK}}M, g^r)$. Once $UGen$ has been executed for users U_i and U_j, then sending a message between the users requires user U_i calling $AEnc_{PK_{LK}}$, L/T calling $\Gamma_{K'_{U_j}}$, and user U_j calling $ADec_{K_{U_j}}$. The encryption scheme $\mathcal{E}$ is correct because $ADec_{K_{U_j}}(\Gamma_{K'_{U_j}}(AEnc_{PK_{LK}}(m))) = m$.

The encryption scheme $\mathcal{E}$ is secure if it retains the same level of security as the standard El Gamal scheme against all adversaries $\mathcal{A}$, and if L/T cannot distinguish between encryptions of two messages even with access to multiple proxy keys.

Theorem 1 Let $\mathcal{E} = (IGen, UGen, AEnc, ADec, \Gamma)$ be the CMLP encryption scheme. $\mathcal{E}$ is CPA (chosen-plaintext attack) secure against L/T and any adversary $\mathcal{A}$. The modification to the proxy encryption scheme over SELS does not affect the security of the scheme. The proof of Theorem 1, therefore, remains the same as that provided in SELS. However, it is provided in Appendix B for the sake of completeness.

In practice asymmetric encryption of bulk messages is costly and, therefore, we employ a hybrid encryption approach where the bulk message is encrypted with a symmetric encryption algorithm (e.g. AES) and then $AEnc$ is used to encrypt the symmetric key. We use $E_k(m)$ to denote the symmetric encryption of message m with key k.

3.2 The CMLP

We assume that all entities, namely, the users, LM and L/T have (or can obtain and trust) each other's public-key certificates for encryption of user subscription e-mails and for signature verification of all e-mails (e.g., PGP certificates or those from an external PKI). We distinguish CMLP keys from external PKI keys by placing a *bar* on top of the PGP/external PKI keys. $Enc_{\overline{PK_i}}$(m) denotes the encryption of message m with public-key $\overline{PK_i}$, and $Sig_{\overline{K_i}}$(m) denotes a signed message (i.e., the message m along with its signature) using private key $\overline{K_i}$. We use El Gamal for encryption and RSA for signatures.

3.2.1 *List Creation*

To create a new list L, LM and L/T execute the following steps:

1. LM begins the execution of $IGen$ and generates parameters $(g, p, q, K_{LM}, g^{K_{LM}})$, and associates the key pair (K_{LM}, PK_{LM}) with the list. LM then makes the key PK_{LM} public.

2. LM then sends L/T a message with the values g, p, and q, and the new list ID L. Formally, $LM \longrightarrow L/T$: $Sig_{\overline{K_{LM}}}$("Create" List L, g, p, q).

3. L/T then continues the execution of $IGen$ by choosing a new private key $K_{L/T}$, computing public keys $PK_{L/T} = g^{K_{L/T}}$ and associating the key pair with the list. L/T then makes the key $PK_{L/T}$ public.

4. Both LM and L/T implicitly agree that the sum $K_{LK} = K_{LM} + K_{L/T}$ (mod q) is the *list key* but neither knows its value since neither knows the other's private key. They complete the execution of $IGen$ by computing $PK_{LK} = PK_{L/T}.PK_{LM} = g^{K_{L/T}}.g^{K_{LM}} = g^{K_{LK}}$ and making it public.

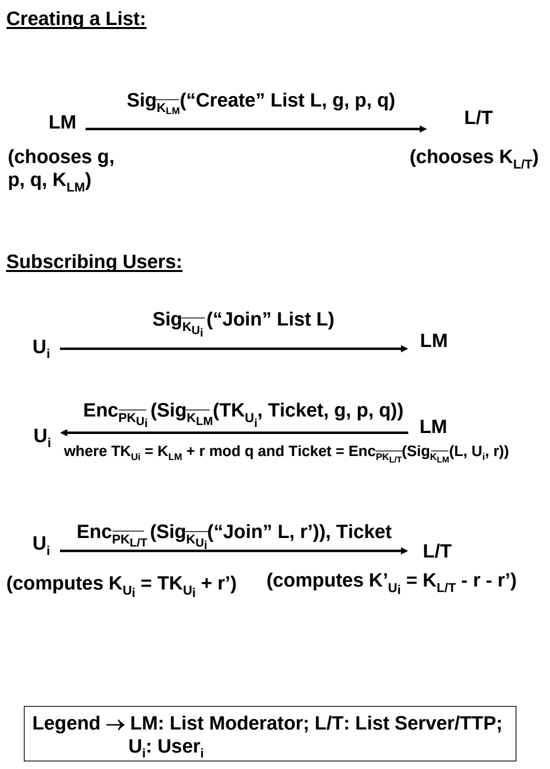

Creating a List:
Sig_KLM("Create" List L, g, p, q)
LM
L/T
(chooses g, p, q, K_LM)
(chooses K_L/T)
Subscribing Users:
Sig_KUi("Join" List L)
U_i
LM
Enc_PKUi(Sig_KLM(TK_Ui, Ticket, g, p, q))
U_i
LM
where TK_Ui = K_LM + r mod q and Ticket = Enc_PKL/T(Sig_KLM(L, U_i, r))
Enc_PKL/T(Sig_KUi("Join" L, r')), Ticket
U_i
L/T
(computes K_Ui = TK_Ui + r')
(computes K'_Ui = K_L/T - r - r')
Legend → LM: List Moderator; L/T: List Server/TTP; U_i: User_i

Figure 1: List Creation and User Subscription

The list is now ready for subscription.

3.2.2 *Subscribing Users*

To subscribe user U_i to list L, U_i, LM and L/T execute the following steps:

1. U_i sends a signed "join" request to LM. Formally, $U_i \longrightarrow LM$: $Sig_{\overline{K_{U_i}}}$("Join" List L).

2. LM authenticates U_i and begins the execution of $UGen$ by generating a random value r, a temporary key $TK_{U_i} = K_{LM} + r$ mod q for U_i, and a ticket encrypted with L/T's public-key containing the value r.

3. LM then sends the values g, p, and q, the temporary key, and the ticket to U_i. Formally, $LM \longrightarrow U_i$: $Enc_{\overline{PK_{U_i}}}$ $(Sig_{\overline{K_{LM}}}(TK_{U_i}$, Ticket, $g, p, q))$ where $TK_{U_i} = K_{LM} + r$ mod q and Ticket = $Enc_{\overline{PK_{L/T}}}$ $(Sig_{\overline{K_{LM}}}(L, U_i, r))$.

4. On receiving this message from LM, U_i generates a random value r' and computes his private key $K_{U_i} = TK_{U_i} + r'$ mod q.

5. U_i then sends the value r' to L/T encrypted with $\overline{PK_{L/T}}$ along with the ticket received from LM. Formally, $U_i \longrightarrow L/T$: $Enc_{\overline{PK_{L/T}}}(Sig_{\overline{K_{U_i}}}$("Join" L, r')), Ticket.

6. L/T authenticates the ticket via LM's signature, obtains r and r' from this message, and computes the proxy key $K'_{U_i} = K_{L/T}$ - r - r' mod q.

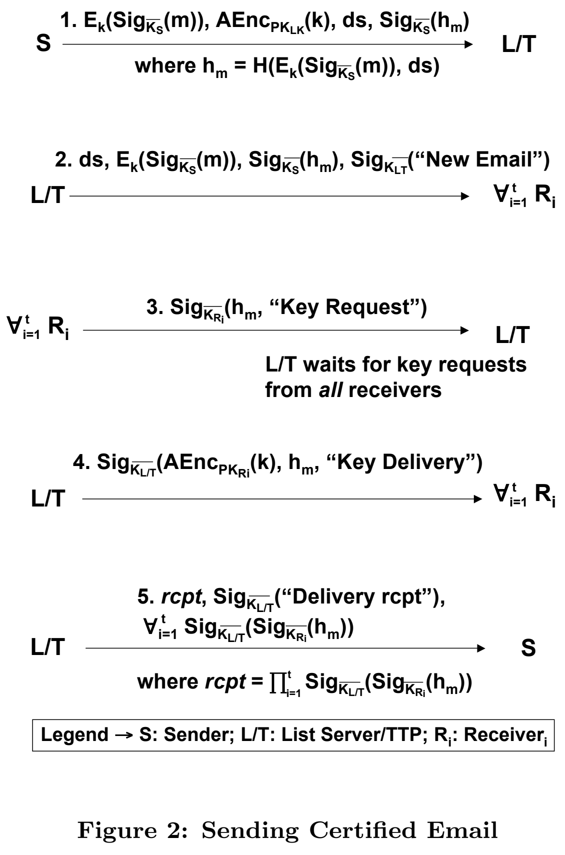
This completes the execution of $UGen$. Note that the sum of keys K_{U_i} and K'_{U_i} is also K_{LK} (the list key) and that neither LM nor L/T knows the user's private key K_{U_i}.

3.2.3 Sending Certified Email

The CMLP protocol for sending certified email on a mailing list is a five-step protocol. In the first step the sender sends a signed and encrypted email message along with a cleartext description to L/T. In the second step L/T forwards the message to all, say t, list subscribers. In the third step each receiver sends a signed key request to L/T to obtain a transformation for the email message. Depending on the choice of strong or weak fairness, L/T waits for requests from either all receivers or from any one receiver. In the fourth step L/T transforms the message key for each receiver and sends it to that receiver. In the fifth step L/T sends a delivery receipt to the sender.

The protocol binds the sender and all receivers to one email message as follows. In the first step, sender computes the hash of the encrypted message and its description. This hash is signed and included in every message and enables the sender and receivers to verify that they are dealing with the same message.

In more detail, the protocol is specified in Figure 2 and runs as follows.



S
1. $E_k(Sig_{\overline{K_S}}(m))$, $AEnc_{PK_{LK}}(k)$, ds, $Sig_{\overline{K_S}}(h_m)$
where $h_m = H(E_k(Sig_{\overline{K_S}}(m))$, ds)
L/T
2. ds, $E_k(Sig_{\overline{K_S}}(m))$, $Sig_{\overline{K_S}}(h_m)$, $Sig_{\overline{K_{L/T}}}$("New Email")
L/T
$\forall^t_{i=1}$ R_i
$\forall^t_{i=1}$ R_i
3. $Sig_{\overline{K_{R_i}}}(h_m$, "Key Request")
L/T
L/T waits for key requests from *all* receivers
4. $Sig_{\overline{K_{L/T}}}(AEnc_{PK_{R_i}}(k)$, h_m, "Key Delivery")
L/T
$\forall^t_{i=1}$ R_i
5. *rcpt*, $Sig_{\overline{K_{L/T}}}$("Delivery rcpt"),
$\forall^t_{i=1}$ $Sig_{\overline{K_{L/T}}}(Sig_{\overline{K_{R_i}}}(h_m))$
L/T
S
where *rcpt* = $\prod^t_{i=1} Sig_{\overline{K_{L/T}}}(Sig_{\overline{K_{R_i}}}(h_m))$
Legend → S: Sender; L/T: List Server/TTP; R_i: Receiver$_i$

Figure 2: Sending Certified Email

Step 1: Sender S performs the following operations:

1. S signs the message m with private key $\overline{K_S}$. That is, S computes $Sig_{\overline{K_S}}(m)$. S also generates a description ds, which is a header asking receivers to read the certified email, perhaps explaining the email contents.

2. S first generates a random key k suitable for E and then encrypts the above signed message. That is, S computes $menc = E_k(Sig_{\overline{K_S}}(m))$.

3. S then computes a hash of the encrypted message and the description. Specifically, S computes $h_m = H(menc, ds)$ where $H()$ is a collision resistant hash function such as SHA. S then signs h_m with private key $\overline{K_S}$.

4. S then computes the asymmetric encryption of key k using its public key PK_{LK}; i.e., it computes $AEnc_{PK_{LK}}(k)$.

5. (*Message 1*) S sends to L/T:
$E_k(Sig_{\overline{K_S}}(m))$, $AEnc_{PK_{LK}}(k)$, ds, $Sig_{\overline{K_S}}(h_m)$.

Step 2: L/T performs the following operations:

1. L/T verifies S's signature on h_m and compares it to the re-computed hash from the received message. If the match does not succeed L/T aborts the protocol.

2. (*Message 2*) $\forall^t_{i=1}$ L/T sends to every list subscriber R_i: $E_k(Sig_{\overline{K_S}}(m))$, ds, $Sig_{\overline{K_S}}(h_m)$,$Sig_{\overline{K_{L/T}}}$("*NewE−mail*")

Step 3: Every receiver R_i performs the following operations:

1. R_i verifies L/T's signature on "New Email" and S's signature on h_m and compares it to the re-computed hash from the received message. If the match does not succeed R_i aborts the protocol.

2. If R_i decides that it wants to read the message (e.g. based on the description ds), it signs h_m with its private key $\overline{K_{R_i}}$.

3. (*Message 3*) $\forall^t_{i=1}$ R_i sends a key request to L/T:
$Sig_{\overline{K_{R_i}}}(h_m$, "*KeyRequest*").

Steps 4 and 5 for strong fairness: L/T performs the following operations:

1. L/T waits for Message 3 from all receivers (in practice, this wait would be time bound).

2. L/T verifies the receivers' signatures on Message 3 and compares h_m from the message to that computed in Step 2. If any of the signatures or matches fail, L/T aborts the protocol.

3. For every receiver R_i, L/T transforms the encrypted symmetric key. That is, L/T computes $AEnc_{PK_{R_i}}(k)$ = $\Gamma_{K'_{R_i}}(AEnc_{PK_{LK}}(k))$.

4. (*Message 4*) $\forall^t_{i=1}$ L/T sends the key to every receiver R_i: $Sig_{\overline{K_{L/T}}}(AEnc_{PK_{R_i}}(k), h_m$, "*KeyDelivery*")

   On receiving this message R_i decrypts k with private key K_{R_i}, uses k to decrypt the email message, and verifies the sender's signature on the email. R_i also recomputes h_m by first re-encrypting the decrypted message with k and verifying that that the hash matches the key request that it had sent to L/T.

5. L/T co-signs every received Message 3 and then computes a condensed signature from these co-signatures [25]. That is, L/T computes $rcpt = \prod_{i=1}^{t} \sigma_i \pmod{n}$ where $\sigma_i = Sig_{\overline{K_{L/T}}}(Sig_{\overline{K_{R_i}}}(h_m))$ and n is the RSA modulus of L/T's signature key.

6. (*Message 5*) L/T sends the delivery receipt to the sender S: $rcpt$, $Sig_{\overline{K_{L/T}}}(\text{“}DeliveryReceipt\text{”})$, $\forall_{i=1}^{t}\ Sig_{\overline{K_{L/T}}}(Sig_{\overline{K_{R_i}}}(h_m))$

   On receiving this message the sender checks the condensed signature by multiplying the hashes of all input messages (i.e. receiver signatures) and verifying that $rcpt^{PK_{L/T}} = \prod_{i=1}^{t} Sig_{\overline{K_{R_i}}}(h_m)$. If the sender wishes to do so, it can verify each individual receiver's signature as well. However, this is not necessary as the sender "trusts" L/T to ensure fairness.

Steps 4 and 5 for weak fairness: L/T performs identical operations to those for strong fairness except that L/T does not wait for all key requests from the receivers; instead, for every key request that it gets from a given receiver L/T transforms k and sends it to that receiver, and sends a signed delivery receipt to the sender.

1. L/T verifies the signature on Message 3 from Receiver R_i and compares h_m from the message to that computed in Step 2. If the signature or match fails, L/T aborts the protocol.

2. L/T computes $AEnc_{PK_{R_i}}(k) = \Gamma_{K'_{R_i}}(AEnc_{PK_{LK}}(k))$.

3. (*Message 4*) L/T sends the key to receiver R_i: $Sig_{\overline{K_{L/T}}}(AEnc_{PK_{R_i}}(k), h_m, \text{“}KeyDelivery\text{”})$

   On receiving this message R_i decrypts k with private key K_{R_i}, uses k to decrypt the email message, and verifies the sender's signature on the email. R_i also recomputes h_m by first re-encrypting the decrypted message with k and then verifying that the hash matches the key request it had sent to L/T.

4. L/T co-signs received Message 3; i.e., L/T computes $rcpt = Sig_{\overline{K_{L/T}}}(Sig_{\overline{K_{R_i}}}(h_m), \text{“}DeliveryReceipt\text{”})$.

5. (*Message 5*) L/T sends the delivery receipt to the sender S: $rcpt$.

   On receiving this message the sender verifies L/T's signature on the receipt.

Separating LS and TTP. For simplicity we combine LS and TTP in the CMLP protocol. However, this is not necessary for the protocol's correctness. If they were separated, LS would execute functions for list creation, user subscription (and associated key management), and forwarding emails to list subscriber (both with and without transformation). The TTP would be placed between list subscribers and LS and would execute functions for certified email delivery. Sender S would send Message 1 to TTP who would then ask LS to forward Message 2 to all receivers. All receivers would send a key request (Message 3) to the TTP. The TTP would then ask LS to transform and send the key to the receivers (Message 4) and would send delivery receipts to S (Message 5). In this case both TTP and LS would be trusted entities (though neither would be able to read messages) and have their own signature keys for which verification keys would be made available to list subscribers.

3.3 Analysis: Satisfying Requirements

In this section we analyse our protocol and informally show that the protocol satisfies the requirements outlined in Section 2. In the next section we discuss formal specification and verification of the protocol using Proverif.

Strong Fairness. Strong fairness requires that a receiver of a given list reads a message m if and only if all list subscribers get m and the sender gets a delivery receipt from every receiver of the list. In the protocol any receiver R_i must obtain k from L/T in order to read m. The protocol ensures that (1) L/T delivers k to R_i only after it receives key requests from all receivers, (2) L/T delivers k to all receivers, and (3) L/T also sends a signed delivery receipt $rcpt$ along with signed key requests from all receivers to S. S can reveal m, $rcpt$, and signed key requests to show that all receivers have read the message.

Weak Fairness. Weak fairness requires that a receiver of a given list reads a message m if and only if the sender gets a delivery receipt from that receiver. The analysis for this requirement is analogous to that for strong fairness.

Confidentiality. The confidentiality requirement for CMLPs is that only the sender and receivers should be able to read messages. In particular, this implies that L/T should not be able to read messages. The protocol ensures this by using the CMLP proxy encryption scheme, which encrypts the symmetric key k with PK_{LK} and enables L/T to transform this such that it is encrypted with the receivers' public keys. This prevents all users that are not part of the list and L/T from accessing message cleartext.

Non-repudiation of origin. This requires that once a receiver gets a message m, the sender should not be able to deny that it originated the message. The protocol ensures this by having the sender digitally sign the message. The receiver can then produce the message and the sender's signature to show that the sender originated the message.

Non-repudiation of receipt. This requires that once the sender obtains a sending receipt from a given receiver, that receiver should not be able to deny having received the message. The protocol ensures this by having the receiver sign h_m, which is co-signed by L/T and sent to the sender as the delivery receipt. When the delivery receipt is sent, L/T also transforms the message key and sends it to the receiver. The sender can produce the message m, key k, $rcpt$, and the receiver's signature on h_m to show that the receiver received the message.

Authentication and Integrity. The protocol ensures this by having the sender sign the outgoing message, the receiver sign the key request, and L/T sign the key delivery and the delivery receipt. The signatures allow message recipients to verify the senders' identities and to ensure that the messages have not been modified in transit.

4. DISCUSSION

In this section we analyze our protocol in terms of user overhead, discuss the proxy encryption scheme used, and examples where the CMLP protocol can be used.

4.1 Assessing User Overhead

An important feature of the CMLP protocol is that it retains the ease of MLs while providing certified delivery. The bulk of the key management overhead is endured by L/T and each list subscriber only has to manage a El Gamal public-private key pair and has to make its signature verification key available. This key pair is not affected by the join and leave of other users and, therefore, there are no re-keying costs associated with list dynamics (i.e., join and leave of subscribers). To send a certified email, the sender simply sends an email encrypted with a symmetric key k (and k encrypted with public key PK_{LK}) to L/T who then handles all tasks and returns a delivery receipt for all receivers to the sender. Unlike previous inline fair non-repudiation protocols [23] where the sender has to send a separate message for releasing k, in CMLP the L/T takes care of releasing k to the recipients after it obtains delivery receipts from them. This is possible because L/T can automatically transform messages for any recipient that sends in a receipt using proxy encryption while in [23] the sender had to compute a group encrypted messages specifically for the users that sent in the receipt. Furthermore, the verification of delivery receipts by the sender is greatly simplified in CMLP via the use of condensed signatures that require the sender to perform only one exponentiation. On the recipient side, each receiver only has to interact with L/T by sending a key request and obtaining a key delivery in return.

4.2 Confidentiality via Proxy Encryption

To satisfy our requirement of confidentiality we use a proxy encryption scheme. This scheme ensures that only the sender and receivers can read email messages; i.e., even L/T cannot read the messages. Such applications of proxy encryption to minimize trust liability in servers (L/T in this case) have previously been explored in [6, 21]. The construction of the proxy transformation function Γ used by the CMLP encryption scheme can be viewed as a two-step threshold decryption process with the decryption key shared between L/T and each subscriber. However, like [20] (who use a similar construction for unidirectional ElGamal proxy encryption) we identify the construction as a proxy encryption scheme because the semantics of the construction convert a message encrypted with one public key (PK_{LK}) into that encrypted with another public key (PK_{R_i}) for some receiver i without revealing the message or any key that can be used to decrypt the message. This is precisely how proxy encryption has been defined in the literature [10, 20].

4.3 Examples of CMLP Use

We now provide two examples one where the strong fairness guarantees of the CMLP protocol is needed and one where weak fairness is needed:

Multi-party negotiations. Any official multi-party document exchange protocol (e.g., for contract-signing, establishing resource-sharing agreements in dynamic coalitions, etc.) would, in practice, require multi-round negotiations among the participating entities before final agreement is reached. We argue that mailing lists provide an easy to use messaging infrastructure for such negotiations and, in fact, is probably used today albeit without any delivery guarantees. Strong fairness in this setting would ensure that no subset of parties can unduly influence the negotiation process as either all parties receive a given message or no one does. Furthermore, non-repudiation of both origin and receipt are important to prevent false denial and false repudiation of proposed and negotiated commitments. In contract-signing the final step would involve each party signing a known document such that either all parties get the signature or no one does. The strong fairness guarantee of the CMLP protocol can be used to ensure this (with the key encrypting the sending party's signature being released only when all receiving parties are willing to give a receipt for the signature); though there are other techniques that provide this as well; e.g. [18].

Information Dissemination. In many workplaces official announcements are sent via mailing lists. However, in the absence of certified delivery, important announcements may not reach recipients or their delivery might be repudiated. For example, consider an announcement about new Visa regulations in a University campus that affects a significant number of international student, faculty, and staff. Such an announcement would typically be sent over a mailing list of international people maintained by the office of international education services. Using weak fairness would enable the office to ensure that all members of the list received the message and that no member can repudiate the fact that they did.

5. FORMAL VERIFICATION

In this section we outline the formalization and verification of the fairness property of our certified mailing list protocol with details provided in the Appendix A. The proxy encryption scheme used by the CMLP protocol and its accompanying proof presented in SELS provides confidentiality. For fairness we use an automatic protocol verification tool, Proverif [2] and obtain results similar to those obtained by Abadi and Blanchet in their verification (using Proverif) of a two-party certified email protocol [1]. We formally specify the protocol, state the propositions for weak fairness (we have not yet undertaken the proof for strong fairness), and show how the tool was used to prove these propositions.

The Verification Tool.
Proverif is a sound verification tool that requires protocols to be expressed in a formal language. The tool then translates the expression into a set of Horn clauses and uses a resolution-based solving algorithm to determine properties of the protocol. An important capability for verifying our protocol is that Proverif can serve for establishing correspondence assertions using specified events. These events allow for establishing assertions of the form, "if $\mathbf{end}(M)$ has been executed then $\mathbf{begin}(M)$ must have been executed" (where $\mathbf{end}(M)$ and $\mathbf{begin}(M)$ denote two events). We use such assertions to show that a list subscriber will be able to receive an email message if and only if the sender gets a corresponding receipt for that message.

The input language and details in the internal representation are provided in [1]. Below we briefly discuss the proof engine and correspondence assertions with additional details in [1], [2].

Proof Engine

The verifier uses a resolution-based solving algorithm to determine the properties of the protocol. It implements a function $solve_{P,Init}(F)$ that takes as parameters the protocol P, the initial knowledge of the adversary $Init$, and a fact F, and returns a set of Horn clauses. After translating the protocol into a set of Horn clauses $\mathcal{C}$, the function saturates this set using a resolution-based algorithm, and finally determines what is derivable. If F' is an instance of F, $\mathcal{C}_b$ is a set of closed facts $begin(p)$, then the tool can show the fact that F' is derivable from $\mathcal{C} \cup \mathcal{C}_b$ if and only if there exists a clause $F \wedge ... \wedge F_n \rightarrow F_0$ in $solve_{P,Init}(F)$ and a substitution σ such that $F' = \sigma F_0$ and σF_1, ..., σF_n are derivable from $\mathcal{C} \cup \mathcal{C}_b$. Values of $solve_{P,Init}(F)$ other than ϕ indicate which begin events must be executed to prove F'.

Correspondence Assertions

The meaning of the correspondence assertion specification are as follows [1]:

Definition 1 (Correspondence) Let P be a closed process and N, $M_{i,j}$ for $i \in \{1, ..., n\}$ and $j \in \{1, ..., l_i\}$ be terms whose free names are among the free names of P. The process P satisfies the correspondence assertion $\textbf{end}(N) \mapsto \bigvee_{i=1}^{n} \textbf{begin}(M_{i1})$, ..., $\textbf{begin}(M_{il_i})$ with respect to $Init$-adversaries if and only if, for any $Init$-adversary Q, for any σ defined on the variables of N, if $\textbf{end}(\sigma N)$ is executed in some reduction trace of $P|Q$, then there exists $i \in \{1, ..., n\}$ such that we can extend σ so that for $k \in \{1, ..., l_i\}$, $\textbf{begin}(\sigma M_{i,k})$ is executed in the trace as well.

The following theorem provides a method for proving these correspondence assertions with Proverif.

Theorem 1 (Correspondence) Let P be a closed process and N, $M_{i,j}$ for $i \in \{1, ..., n\}$ and $j \in \{1, ..., l_i\}$ be terms whose free names are among the free names of P. Let $p_{i,j}$ be the patterns obtained by replacing each name a with the corresponding pattern $a[]$ in the terms N, M_{ij} respectively. Assume that for all rules R in $solve_{P,Init}(end(p))$, there exist $i \in \{1, ..., n\}$, σ', and H such that $R = H \wedge$ begin($\sigma' p_{i1}$) $\wedge ... \wedge$ begin($\sigma' p_{il_i}$) $\rightarrow$ end($\sigma' p$). Then P satisfies the correspondence assertion $\textbf{end}(N) \mapsto \bigvee_{i=1}^{n} \textbf{begin}(M_{i1})$, ..., $\textbf{begin}(M_{il_i})$ with respect to $Init$-adversaries.

Verification of Weak Fairness

In Appendix A we provide a formalization of the CMLP protocol in the input language of the verification tool.

The correctness property for weak fairness is that a receiver gets message m if and only if the sender gets a receipt. The proof should be such that, if the sender goes to a judge, then the judge can definitely state that the receiver has received the message. This property holds only when the delivery of messages is guaranteed on the channels between the sender, judge, and receivers.

The requirement for weak fairness translates into the following two propositions; the first in which the receiver attempts to cheat and the second in which the sender attempts to cheat. We show how the tool is used to provide proofs for these propositions in the Appendix A.

Proposition 1. Assume that messages from L/T sent to the sender and from sender sent to the judge reach their destination. If the receiver has received message m, then the judge says that the receiver has received m.

Proposition 2. Assume that messages sent from L/T reach their destination. If the judge says that the receiver has received m, then the receiver has received m.

6. CONCLUSIONS

In this paper we presented a novel Certified Mailing-list Protocol (CMLP) that provides fair delivery, confidentiality, non-repudiation of origin and receipt, authentication, and integrity. We formally specified and verified the weak fairness property of the protocol with Proverif. In the future we will formally verify the strong fairness property of the protocol and implement a prototype.

7. ACKNOWLEDGEMENTS

We would like to thank Rakesh Bobba and the anonymous reviewers for helpful comments and suggestions. This work was funded by the Office of Naval Research under contract number N00014-04-1-0562.

8. REFERENCES

[1] M. Abadi and B. Blanchet, "Computer-Assisted Verification of a Protocol for Certified Email", in the 10th International Symposium (SAS'03), San Diego, California, June 2003.

[2] M. Abadi and B. Blanchet, "Analyzing Security Protocols with Secrecy Types and Logic Programs", in *Journal of the ACM, 52(1):102-146*, January 2005.

[3] M. Abadi, N. Glew, B. Horne, and B. Pinkas, "Certified email with a light on-line trusted third party: Design and implementation", in the Eleventh International World Wide Web Conference, New York, 2002.

[4] N. Asokan, M. Schunter, and M. Waidner, "Optimistic Protocols for Fair Exchange", in Proceedings of 4th ACM Conference on Computer and Communications Security, Zurich, April 1997.

[5] N. Asokan, Matthias Schunter, and Michael Waidner, "Optimistic protocols for multi-party fair exchange", Research Report RZ 2892 (90840), IBM Research, December 1996.

[6] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage, in Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS), February 2005.

[7] G. Ateniese, B. de Medeiros, and M. Goodrich, "Tricert: Distributed certified e-mail schemes", in the Network and Distributed System Security Symposium, San Diego, CA, February 2000.

[8] F. Bao, R. Deng, K. Q. Nguyen, and V. Vardharajan, "Multi-party fair exchange with an off-line trusted neutral party", in DEXA'99 Workshop on Electronic Commerce and Security, Firenze, Italy, September 1999.

[9] S. Berchtold, A. Biliris, and E. Panagos, "SaveMe: a system for archiving electronic documents using messaging groupware", in the international joint conference on Work activities coordination and collaboration, California, United States, 1999.

[10] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography", in Eurocrypt'98, LNCS 1403, Springer-Verlag, 1998.

[11] C. Blundo, S. Cimato, and R. D. Prisco, "Certified Email: Design and Implementation of a New Optimistic

Protocol", in proceedings of the Eighth IEEE International Symposium on Computers and Communications, June 30 - July 03, Turkey, 2003.

[12] D. Boneh, "The Decision Diffie-Hellman Problem", in proceedings of the International Symposium on Algorithmic Number Theory (ANTS'98), 1998.

[13] Catalist, the official catalog of LISTSERV lists, *http://www.lsoft.com/catalist.html.*

[14] R. H. Deng, L. Gong, A. A. Lazar, and W. Wang, "Practical protocols for certified electronic mail", in the *Journal of Network and Systems Management*, 3(4), 1996.

[15] J. Ferrer-Gomila, M. Payeras-Capella, and L. Huguet-Rotger, "A realistic protocol for multi-party certified electronic mail", in proceedings of Information Security Conference, Sao Paulo, Brazil, September 2002.

[16] M. Franklin and M. Reiter, "Fair exchange with a semi-trusted third party", in the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1997.

[17] M. Franklin and G. Tsudik, "Secure group barter: multi-party fair exchange with semi-trusted neutral parties", in Financial Cryptography, 1998.

[18] J. A. Garay and P. MacKenzie, P, "Abuse-free multi-party contract signing", in Distributed Computing (DISC '99), Bratislava, Slovak Rep., 27-29 Sep., vol. 1693 of Lecture Notes in Computer Science, 1999.

[19] C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development", in Proceedings of the 2004 ACM conference on Computer supported cooperative work, Illinois, USA, 2004.

[20] A. Ivan and Y. Dodis, "Proxy Cryptography Revisited", in Proceedings of the Network and Distributed System Security Symposium (NDSS), February 2003.

[21] H. Khurana, A. Slagell, and R. Bonilla, "SELS: A Secure E-mail List Service", n the Security Track of the ACM Symposium on Applied Computing (SAC), March 2005.

[22] E. Kirda, P. Fenkam, G. Reif, H. Gall, "A service architecture for mobile teamwork", in Proceedings of the 14th international conference on Software engineering and knowledge engineering, Ischia, Italy, 2002.

[23] S. Kremer and O. Markowitch, "A multi-party non-repudiation protocol", in proceedings of 15th IFIP International Information Security Conference, Beijing, China, August 2000.

[24] O. Markowitch and S. Kremer, "A multi-party optimistic non-repudiation protocol", in proceedings of 3rd International Conference on Information Security and Cryptology, Seoul, Korea, December 2000.

[25] E. Mykletun, M. Narasimha, G. Tsudik, "Authentication and Integrity in Outsourced Databases", in ISOC Symposium on Network and Distributed Systems Security, 2004.

[26] C. Weng and J. H. Gennari, "Asynchronous collaborative writing through annotations", in Proceedings of the 2004 ACM conference on Computer supported cooperative work, Illinois, USA, 2004.

[27] J. Zhou, "On the Security of a Multi-Party Certified Email Protocol", in proceedings of the International Conference on Information and Communications Security, Malaga, Spain, October 2004.

[28] J. Zhou and D. Gollmann, "A fair non-repudiation protocol", in proceedings of the IEEE Symposium on Research in Security and Privacy, 1996.

APPENDIX

A. FORMAL VERIFICATION

In this section we present details of the formalization and verification of our certified mailing list protocol. We formally specify the protocol, state the propositions for weak fairness, and show how the tool was used to prove these propositions.

A.1 Formalizing the Protocol

The following is the coding of the CMLP protocol in the verifier's input language with the steps of the protocol identified in the comments. This code represents the situation in which all principals behave honestly. We modify the situation in the next section to verify our propositions.

```
  (* Public key cryptography *)
fun pk/1.
fun encrypt/2.
reduc decrypt(encrypt(x,pk(y)),y) = x.

(* Signatures *)
fun sign/2.
reduc getmess(sign(m,k)) = m.
reduc checksign(sign(m,k),pk(k)) = m.

(* Host names *)
fun host/1.
reduc getkey(host(x)) = x.

(* Proxy Key: LT tranforms a message m encrypted with pkY into
a message encrypted with pkZ using the proxy key associated with
pkZ *)
reduc transform(encrypt(m,pk(y)),host(pk(y)),host(pk(w)),
      lookupproxykey(host(pk(y)),host(pk(w)))) = encrypt(m,pk(w)).

(* hash function *)
fun hash/1.

(* Shared-key cryptography *)
fun sencrypt/2.
reduc sdecrypt(sencrypt(x,y),y) = x.

(* Build a message*)
private fun Message/1.

(* Free names (public and private constants) *)
(* Signature keys *)
private free sigSKey,sigLTKey,sigRKey.
(* Decryption keys *)
private free dSKey,dRKey,dLTKey.
(* channels *)
free c,LTchannel,Schannel,Rchannel.
(* Constants to identify messages *)
data Give/0. data NewEmail/0. data KeyRequest/0.
data KeyDelivery/0.data DeliveryReceipt/0. data Delivered/0.
```

```
let processS =
  (* Step 1.1: Build a message to send and sign it *)
  new text; new desc; new skey;
  let mesg = Message(text) in
  event beginSmesg(mesg);
  let signedMesg = sign((Sname,desc,mesg),sigSKey) in
  (* Step 1.2 *)
  let sencM1 = sencrypt((mesg,signedMesg),skey) in
  (* Step 1.3 *)
  let hashM1 = hash((Sname,desc,sencM1)) in
  (* Step 1.4 *)
  let aencM1 = encrypt(skey,eLKkey) in
  (* Step 1.5 *)
  let signedHashM1 = sign((hashM1,Give),sigSKey) in
  event beginSGive(Sname,desc,sencM1,aencM1,signedHashM1);
  out(LTchannel, (Sname,desc,sencM1,aencM1,signedHashM1));
  !
  in(Schannel,(LTnameIn,signedM5In));
  (* Step 4/5.5 *)
  let (receipt,aencIn,=DeliveryReceipt,receiver) =
     checksign(signedM5In,verLTKey) in
  let (=hashM1,=KeyRequest,=receiver) = getmess(receipt) in
  event endLTDeliveryReceipt(LTnameIn,signedM5In);
        signedHashM1,signedM5In));
  (* Send delivery receipt copy to other sender processes *)
  out(Schannel,(LTnameIn,signedM5In)).

let processLT =
  in(LTchannel, (SnameIn,descM2,sencM2,aencIn,signedHashM2));
  (* Step 2.1 *)
  let hashInM2 = hash((SnameIn,descM2,sencM2)) in
  let (=hashInM2,=Give) = checksign(signedHashM2,verSKey) in
  (* Step 2.2 *)
  let signedNewEmail = sign((NewEmail,SnameIn),sigLTKey) in
  event beginLTNewMesg(LTname,descM2,sencM2,signedHashM2,
     signedNewEmail);
  event endSGive(SnameIn,descM2,sencM2,aencIn,signedHashM2);
  out(Rchannel,(LTname,descM2,sencM2,signedHashM2,
     signedNewEmail));
  !
  in(LTchannel,signedRM5);
  (* Step 4/5.1 *)
  let (=hashInM2,=KeyRequest,receiver) =
     checksign(signedRM5,verRKey) in
  let pkey = lookupproxykey(LTname,receiver) in
  (* Step 4/5.2 *)
  let aencM4 = transform(aencIn,LTname,receiver,pkey) in
  let signedM4 = sign((aencM4,hashInM2,KeyDelivery,receiver),
     sigLTKey) in
  let signedM5 = sign((signedRM5,aencM4,DeliveryReceipt,receiver),
     sigLTKey) in
  event endRKeyRequest(signedRM5);
  event beginLTKeyDelivery(LTname,signedM4);
  (* Step 4/5.4 *)
  out(Rchannel,(LTname,signedM4));
  event beginLTDeliveryReceipt(LTname,signedM5);
  (* Step 4/5.5 *)
  out(Schannel,(LTname,signedM5)).


let processR =
  in(Rchannel, (LTnameIn,descIn,sencIn,signedHashIn,
     signedNewEmailIn));
  (* Step 3.1 *)
  let (=NewEmail,sender) =
     checksign(signedNewEmailIn,verLTKey) in
  let hashInM3 = hash((sender,descIn,sencIn)) in
  let (=hashInM3) = checksign(signedHashIn,verSKey) in
  (* Step 3.3 *)
  let signedM3 = sign((hashInM3,KeyRequest,Rname),sigRKey) in
  event endLTNewMesg(LTname,descIn,sencIn,signedHashIn,
     signedNewEmailIn);
  event beginRKeyRequest(signedM3);
  out(LTchannel,signedM3);
  !
  in(Rchannel,(LTnameInM4,signedM4In));
  let (aencIn,=hashInM3,=KeyDelivery,=Rname) =
     checksign(signedM4In,verLTKey) in (
  (* Step 4//5.3 *)
  let dKey = decrypt(aencIn,dRKey) in
  let (decmesg,decsign) = sdecrypt(sencIn,dKey) in
  let (=sender,=descIn,=decmesg) =
     checksign(decsign,verSKey) in
  event endLTKeyDelivery(LTnameInM4,signedM4In);
  event endSMesg(decmesg))
  (* Send receipt out for other receiver processes *)
  else out(Rchannel,(LTnameInM4,signedM4In)).

process
  let verSKey = pk(sigSKey) in out(c, verSKey);
  let verLTKey = pk(sigLTKey) in out(c, verLTKey);
  let verRKey = pk(sigRKey) in out(c, verRKey);
  let eSKey = pk(dSKey) in out(c, eSKey);
  let eRKey = pk(dRKey) in out(c, eRKey);
  let eLKkey = pk(dLTKey) in out(c, eLKkey);
  let Sname = host(eSKey) in out(c, Sname);
  let Rname = host(eRKey) in out (c, Rname);
  let LTname = host(eLKkey) in out(c, LTname);
  ((!processS) | (!processLT) | (!processR))
```

The code begins by declaring cryptographic primitives. For example, encrypt is the public-key encryption constructor that takes a public key and a plaintext to return a ciphertext, and decrypt is the corresponding destructor that takes a ciphertext and a secret key to return the plaintext. We assume perfect cryptography so only the secret key can be used to decipher encrypted messages. We similarly define primitives for shared key encryption, hash function, signatures, and proxy transformation. For proxy transformation we do not encode the cryptographic operations; instead, we encode the functionality of enabling LT to transform messages encrypted with one public one into that encrypted with another public key. We also declare a number of constants that appear in messages. These include names of message channels, signature keys, and decryption keys. All declaration with free are accessible to the adversary while those with private free are not.

The processes processS, processR, and processLT represent S, R, and LT respectively. They are composed in the last part of the protocol under process, which also includes computation of public keys and principal names. The public keys and principal names are then revealed on the public channel, c. Since an infinite number of copies of these process are launched, a single process processR suffices to represent all the list subscribers. By including the receiver's name in the key request, LT can ensure that it uniquely transforms the sender's message for every recipient.

The process processS first composes a message to be sent to LT (for forwarding to list subscribers), signs the message with its signing key sigSKey, generates a new shared key skey, and encrypts the signed message with that key. The process then computes and signs the hash, encrypts skey with the list public encryption key eLKkey, and sends the email out on LTchannel thus completing Step 1. The channel always indicates the destination of the message but since it is public the adversary can read the message and send its own messages on the channel. This part of the process also marks two events; one for composing the message and one for sending the email to LT. In the second part of the process when a message is received from LT we use a pattern matching construct: **let** (=hashM1,=KeyRequest,=receiver) = getmess(receipt) **in** ... A pattern $(p_1, p_2, ..., p_n)$ matches a tuple of arity n, when $p_1, ..., p_n$ match the components of the tuple. So, the destructor application of Step 4/5.5 succeeds if and only if the receipt from LT matches the expected key request from the receiver where LT's signature on the receipt was verified in the previous step. Once this match succeeds, processS executes the event endLTDeliveryReceipt concluding that the given receiver has read the email (weak fairness). The process then forwards this delivery receipt to Schannel so that another session of S can get it. We assume that this is a fair execution so that all sessions of S can get delivery receipts from all receivers; i.e., from all list subscribers.

The process processLT, in the first part, accepts a message from the sender after verifying the sender's signature on the message. processLT then forwards the message on Rchannel after signing the sender's name. In the second part, processLT accepts a message from the receiver and verifies that it is a valid key request. The process then transforms the shared key (that was used to encrypt the email message) so that it is encrypted with the receiver's public key and sends the key delivery message on Rchannel. The process also composes a delivery receipt for the sender by co-signing the receiver's key request and sends the delivery receipt on Schannel. As the process executes various steps of the protocol it triggers events to denote success in those steps.

The process processR, in the first part, accepts a message from LT after verifying the signature on the message. The process then verifies that this is a new email by verifying the sender's signature on the encrypted email message. The process then composes a key request by signing the hash and sends the key request on LTchannel. In the second part, the process accepts a message from LT after verifying the signature on the message. processR then verifies that the key delivery corresponds to the key request, decrypts the emails message, and verifies the sender's signature on the plaintext. If the key delivery does not correspond to the key request, the process forwards it on Rchannel so that other sessions of R can have the chance to get it. As the process executes various steps of the protocol it triggers events to denote success in those steps.

A.2 Results: Verification of Weak Fairness

The correctness property for weak fairness is that a receiver R_i gets message m if and only if S gets a receipt. The proof should be such that, if S goes to a judge, then the judge can definitely state that R_i has received the message. This property holds only when the delivery of messages is guaranteed on the channels between S and LT, R and LT, and between S and Judge. The following definitions are from [1].

Definition 2 A message m sent on a channel c reaches its destination if and only if it is eventually received by an input on channel c in the initial process P_0 or a process derived from it. If the adversary receives the message, it reemits the message on channel c.
Definition 3 Fairness Hypothesis. If a reduction step can be infinitely often executed, then it will eventually be executed. If a message m is sent on channel c, and some inputs on channel c reemit it while some don't, then m will eventually be received by an input that does not reemit it.

These definition of message delivery and fairness hypothesis cannot be taken into account by the verifier so the weak fairness property cannot be proved in a fully automatic way. However, a correspondence assertion that constitutes the most important part of the proof can be proven using the verifier. That is, instead of proving that if some event e_1 has been executed then some event e_2 must have been executed, the verifier shows that some event e'_2 must have been executed. A short manual proof is then needed to show that e_2 will be executed after e'_2.

The following process represents the judge.

```
private free JudgeChannel.
let processJudge =
  in(JudgeChannel,(SnameM6,descM6,mesgM6,skeyM6,sencM6,
      aencInM6,signedMesg,signedHashM6,signedM6In));
  (* Verify sender's signature on message and hash *)
  let hashM6 = hash((SnameM6,descM6,sencM6)) in
  let (=SnameM6,=descM6,=mesgM6) =
      checksign(signedMesg,verSKey) in
  let (=hashM6,=Give) = checksign(signedHashM6,verSKey) in
  (* Verify LT's signature on receiver's key request *)
  let (request,=aencInM6,=DeliveryReceipt,receiver) =
      checksign(signedM6In,verLTKey) in
  (* Verify receiver's request *)
  let (=hashM6,=KeyRequest,=receiver) =
      checksign(request,verRKey) in
  (* Verify that the transformation was done correctly; if so, the
judge says that receiver got mesgM6. *)
  let (=mesgM6) = sdecrypt(sencM6,skeyM6) in
  let (=aencInM6) = encrypt(skeyM6,eRKey) in
  event JudgeSays(Delivered,receiver,mesgM6).
```

In this process, the judge receives a certificate from S and verifies it. If the verification succeeds, the judge says that the receiver has received the message. At the end of the process processS, S sends to the judge:

```
out(JudgeChannel,(Sname,desc,mesg,skey,sencM1,aencM1,aencIn,
      signedMesg,signedHashM1,signedM5In);
```

The requirement for weak fairness translates into the following two propositions; the first in which the receiver attempts to cheat (i.e. R_i is included in the adversary) and the second in which the sender attempts to cheat (i.e. S is included in the adversary).

Proposition 1. Assume that messages from LT sent to

S and from S sent to *Judge* reach their destination. If R_i has received message m, then the judge says that R_i has received m.

Here R is included in the adversary; i.e., R tries to get a message without S obtaining the corresponding delivery receipt. The process for R then becomes:

out(c, dRKey); **out**(c, sigRKey); **in**(Rchannel, m); **out**(LTchannel,m); ! **in**(Rchannel,m2); **event** endSMesg(m2).

The modified process now reveals all the information available to R. The adversary can now execute the event endSMesg(m2) after obtaining message $m2$ and sending it on c. Writing P_0 for the resulting process that represents the whole system, the proposition can be more formally stated as:

Proposition 1' Assume that the messages from LT sent on Schannel and from S sent on JudgeChannel reach their destinations. Let $Init = \{$ Sname, Schannel, JudgeChannel, LTname, LTchannel, c, desc $\}$. For any $Init$-adversary Q, if the event endSMesg(Message(M_x)) is executed in a reduction trace of $P_0|Q$ for some term M_x, then the event JudgeSays(Delivered, M_x, Message(M_x)) is executed in all continuations of this trace.

In processLT we have defined an event beginLTDeliveryReceipt(LTname, sign((signedRM5, aencM4, DeliveryReceipt, receiver), sigLTKey))) to note that LT has sent the receipt to S. In processS we have defined an event beginSGive(Sname, desc, sencM1, aencM1, signedHashM1) to note that S is ready to send the message to LT and obtain a receipt.

Automatic part of the proof: The tool was invoked with the query endSMesg(Message(x)), to determine under which conditions an instance of the corresponding event may be executed. The tool then returns a clause after computing $solve_{P_0,Init}(endSMesg(Message(x)))$ of the form:

begin:beginLTDeliveryReceipt(LTname,sign((sign((hash((Sname, desc,sencrypt((Message(p_x),sign((Sname,desc,Message(p_x), sigSKey)),p_k))),KeyRequest,p_q),sigRKey), encrypt(p_k,p_q), DeliveryReceipt,p_q),sigLTKey)) $\wedge$

begin:beginSGive(Sname,desc,sencrypt((Message(p_x),sign((Sname, desc,Message(p_x),sigSKey)),p_k),encrypt(p_k,eLTKey), sign((hash((Sname,desc,sencrypt((Message(p_x), sign((Sname, desc,Message(p_x)), sigSKey)),p_k))),Give),sigSKey)) $\wedge$

$H \rightarrow$ endSMesg(Message(p_x))

for some patterns p_x, p_q and some hypothesis H. So, by Theorem 1 if endSMesg(Message(p_x)) is executed in a trace of P_0—Q then the events

beginLTDeliveryReceipt(LTname,sign((sign((hash((Sname,desc, sencrypt((Message(M_x),sign((Sname,desc,Message(M_x),sigSKey)), M_k))),KeyRequest,M_q),sigRKey),encrypt(M_k,M_q), DeliveryReceipt,M_q),sigLTKey))

beginSGive(Sname,desc,sencrypt((Message(M_x),sign((Sname, desc,Message(M_x),sigSKey)),M_k),encrypt(M_k,eLTKey), sign((hash((Sname,desc,sencrypt((Message(M_x),sign((Sname,desc, Message(M_x)),sigSKey)),M_k))),Give),sigSKey))
are executed in this trace for some terms M_k, M_p, M_q.

Manual part of the proof: Since LT executes beginLTKeyDelivery(LTname, sign((signedRM5, aencM4, DeliveryReceipt, receiver),sigLTKey) as proved above, it will then execute out(Schannel, (LTname, sign((signedRM5, aencM4, DeliveryReceipt, receiver), sigLTKey))). This message will be received by an input on Schannel from P_0 (we assume that messages reach their destination). In fact, this message will be received by a session of S that does not reemit it (by the fairness hypothesis). Such a session successfully checks the signed receipt and sends it to the judge on JudgeChannel. The message will be received by the input on processJudge, which will check it successfully (the check always succeeds when S's check succeeds). processJudge will then execute JudgeSays(Delivered, M_x, Message(M_x)).

Proposition 2. Assume that messages sent from L/T reach their destination. If the judge says that R_i has received m, then R_i has received m.

Here S is included in the adversary as it may try to get the judge to say that R has received a message it does not have; i.e., processS is simply out(c, sigSkey); out(c, dSkey).. Writing P_0 for the resulting process that represents the whole system, the proposition can be more formally stated as:

Proposition 2' Assume that messages from LT reach their destination. Let $Init = \{$Sname, Schannel LTname, LTchannel, JudgeChannel, c, desc $\}$. For any $Init$-adversary Q, if JudgeSays(Delivered, Rname, M_m) is executed in a reduction trace of $P_0|Q$ for some term M_m, then the event endSMesg(Message(M_m)) is executed in all continuations of this trace.

In processR we have defined an event beginRKeyRequest(sign ((hash(Sname,desc,sencM1)),sigRKey)) to note that R has received the encrypted message and wants to obtain the decryption key. In processLT we have defined an event beginLTKeyDelivery(LTname, sign((aencM4,hashInM2,KeyDelivery,receiver),sigLTKey)) to denote that LT sends the key to R.

Automatic part of the proof: We invoked the verifier with the query JudgeSays(Delivered,Rname,m). The tool then computed a set of clauses $solve_{P_0,Init}(JudgeSays(Del-ivered, Rname, m))$ and returns a clause of the form:

begin:beginLTKeyDelivery(LTname,sign((encrypt(p_y,eRKey), hash((p_w,p_z,sencrypt(m,p_y))),KeyDelivery,Rname),sigLTKey)) $\wedge$

begin:beginRKeyRequest(sign((hash((p_w,p_z,sencrypt(m,p_y))), KeyRequest,Rname),sigRKey)) $\wedge$

$H \rightarrow$ end:JudgeSays(Delivered,Rname,m)

for some patterns p_y, p_w, p_z and some hypothesis H. Therefore, by Theorem 1, if the event JudgeSays(Delivered, Rname, M_m) is executed in a reduction trace of P_0—Q for some term M_m, then the events

beginLTKeyDelivery(LTname,sign((encrypt(M_y,eRKey), hash((M_w,M_z,sencrypt(M_m,M_y))),KeyDelivery,Rname),sigLTKey)) beginRKeyRequest(sign((hash((M_w,M_z,sencrypt(M_m,M_y))), KeyRequest,Rname),sigRKey)) are executed in this trace for some terms M_y, M_w, M_z.

Manual part of the proof: After LT executes event beginLTKeyDelivery(LTname, sign((aencM4, hashInM2, KeyDelivery, receiver), sigLTkey)) it will execute out(Rchannel,(LTname, sign ((aencM4, hashInM2, KeyDelivery, receiver), sigLTkey))). This message reaches its destination, so it will be received by an input derived from P_0. Since the value of receiver must correspond, the session of R that receives this message is also the one that executed beginRKeyRequest(sign((hash(Sname, desc, sencM1)), sigRKey)). This session will then execute endSMesg(Message(M_m)).

B. CMLP ENCRYPTION SCHEME

Theorem 1 - Let $\mathcal{E} = (IGen, UGen, AEnc, ADec, \Gamma)$ be the CMLP encryption scheme. $\mathcal{E}$ is CPA secure against the List Server and any Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$, if El Gamal is CPA secure against such adversaries.

Define,

$$Succ_{LT,\mathcal{E}} \overset{def}{=}$$

$$\Pr\left[b = \hat{b} \,\middle|\, \begin{array}{l} (g,p,q,K_{LM},g^{K_{LM}},K_{L/T},g^{K_{L/T}}) \leftarrow IGen(1^k), \\ (K_u,K'_u) \leftarrow UserGen(1^k,K_{L/T},K_{LM}), \\ b \leftarrow \{0,1\}, (m_0,m_1) \leftarrow L/T(g^{K_u},K'_u), \\ \hat{b} \leftarrow L/T(g^{K_u},K'_u,Enc_{g^{K_u}}(m_b)) \end{array} \right]$$

Then $\mathcal{E}$ is CPA (Chosen Plaintext Attack) secure against L/T if $|Succ_{L/T,\mathcal{E}} - \frac{1}{2}|$ is negligible for L/T. A similar formulation can be made for other adversaries with slight notational changes.

Proof: We have two types of adversaries to consider: the L/T and users outside the list. (List subscribers and LM receive emails from the sender, and, therefore, are not adversaries.) We first consider L/T and assume that it can break the CMLP encryption scheme $\mathcal{E}$. Then $|Succ_{L/T,\mathcal{E}} - \frac{1}{2}|$ is non-negligible. Based on L/T's algorithm to break $\mathcal{E}$, we create a probabilistic, polynomial time (PPT) algorithm $\mathcal{B}$ to mount a successful chosen plaintext attack against the standard El Gamal encryption scheme. However, our premise is that El Gamal is CPA secure. This fact will provide the contradiction to our assumption that L/T can mount a successful CPA attack against $\mathcal{E}$.

We note that L/T has knowledge of multiple proxy keys. Intuitively, we see that L/T cannot decrypt any e-mails with these keys, and cannot gain any knowledge of the list key K_{LK} because its view of the proxy keys can be made consistent with any value of K_{LK}. However, we formally prove that L/T cannot break $\mathcal{E}$ without breaking El Gamal. To prove this we use the idea that an adversary can simulate the role of LM and subscribers since all that L/T receives from them are randomly chosen integers.

An oracle executes $IGen$ and $UGen$ with L/T to generate private and proxy keys K_{LM}, $K_{L/T}$, K_u, and K'_u. The oracle then gives $\mathcal{B}$ keys K_{LM} and g^{K_u} (where g^{K_u} is the El Gamal challenge public key). $\mathcal{B}$ then simulates the subscribing and unsubscribing of (polynomial many) users for L/T by repeated execution of $UGen$ and sending of "unsubscribe" messages. L/T now chooses two messages (m_0, m_1) to challenge the security of our encryption scheme $\mathcal{E}$. $\mathcal{B}$ considers these messages as the challenge to standard El Gamal and receives the challenge $Enc_{g^{K_u}}(m_b)$ from a left-right oracle. This $Enc_{g^{K_u}}(m_b) = AEnc_{g^{K_u}}(m_b)$ challenge is forwarded to L/T who has a distinguisher able to determine b with probability greater than .5 by the assumption that $|Succ_{L/T,\mathcal{E}} - \frac{1}{2}|$ is non-negligible. However, this is a contradiction to the assumption that El Gamal is CPA secure. Therefore, if El Gamal is CPA secure, our encryption system $\mathcal{E}$ is CPA secure against L/T.

We now consider adversaries outside of the list. It is trivial to see that outsiders would have to break El Gamal to decrypt a message. This is because both messages from the sender to L/T and messages from L/T to the receivers are encrypted with valid El Gamal keys which are unknown in part or whole to any outsider. The formal proof is similar to the previous one. The main difference is that an outsider does not participate in the protocol at all. In fact, the algorithm $\mathcal{B}$ simply simulates an entire CMLP system for the adversary giving him access to the communications and to all public-keys. At some point in time the adversary breaks $\mathcal{E}$, and we can show that $\mathcal{B}$ can break El Gamal for the same messages.