Stateless-Recipient Certified E-mail System based on Verifiable Encryption

Giuseppe Ateniese and Cristina Nita-Rotaru

Department of Computer Science
The Johns Hopkins University
{ateniese,crisn}@cs.jhu.edu

Abstract. In this paper we present a certified e-mail system which provides fairness while making use of a TTP only in exceptional circumstances. Our system guarantees that the recipient gets the content of the e-mail if and only if the sender receives an incontestable proof-of-receipt. Our protocol involves two communicating parties, a sender and a recipient, but only the recipient is allowed to misbehave. Therefore, in case of dispute, the sender solicits TTP's arbitration without involving the recipient. This feature makes our protocols very attractive in real-world environments in which recipients would prefer to assume a passive role rather than being actively involved in dispute resolutions caused by malicious senders. In addition, in our protocol, the recipient can be *stateless*, i.e., it does not need to keep state to ensure fairness.

1 Introduction

The Internet has revolutionized the computer and communications world like nothing before and it is today an important global resource for millions of people. With its continue growing, it generated tremendous benefits for the economy and our society. However, the Internet does not provide all the services required by the business communication model such as secure and fair electronic exchange or certified electronic delivery.

A fair electronic exchange protocol ensures that, at the end of the exchange, either each player receives the item it expects or neither part receives any information about the other's item. The classical solution to the fair exchange problem is based on the idea of *gradually* exchanging small parts of the items. However, practical solutions to the problem require a trusted third party (TTP) as arbitrator. More specifically, in *on-line* protocols the trusted party is employed as a delivery channel whereas in *off-line* protocols the trusted party is involved only in case of dispute. On-line protocols require the presence of the TTP in every transaction and, usually, do not provide confidentiality of the items exchanged. In addition, in some cases, the sender receives a receipt signed by the TTP rather than by the original recipient of the message. In off-line protocols, the TTP is invoked only under exceptional circumstances, for example in case of disputes or emergencies.

In a certified e-mail scheme the intended recipient gets the mail content if and only if the mail originator receives an irrefutable proof-of-delivery from the recipient.

In this paper we present a certified e-mail system which implements an off-line protocol that makes use of *verifiable encryption* of digital signatures as building block. A *verifiable encryption* of a digital signature represents a way to encrypt a signature under a designated public key and subsequently prove that the resulting ciphertext indeed contains such a signature.

The rest of the paper is organized as follows. The next section discusses related work done in the areas of fair exchange and certified e-mail protocols. Section 3 outlines the certified e-mail protocol used by our system. We analyze the protocol in Section 4. Finally, we discuss the implementation of the system in Section 5.

2 Related Work

One approach in solving the *fair exchange* problem consists of gradually exchanging information about the items between the two parties. Works in this direction generally rely on the unrealistic assumption that the two parties have equal computational power ([12]) or require many rounds to execute properly ([5]).

Another approach focuses on increasing the overall efficiency by using TTPs. Notable works in this direction are the three-message off-line protocol for certified e-mail presented in [18] and the efficient off-line fair exchange protocols in [1, 2, 7]. The protocol in [1] makes use of verifiable escrow schemes implemented via a *cut and choose* interactive proof. Although expensive, the protocol provides timely termination assuming only resilient channels.

A on-line certified e-mail protocol is presented in[24]. The protocol uses as TTP a number of replicated servers. This has the drawback that each server must be trusted in order to have the protocol working properly. Having only one single compromised server would invalidate the entire scheme. Bahreman and Tygar [6] present an on-line scheme using six messages. The scheme does not address confidentiality from the TTP. An optimal on-line certified e-mail protocol using only four messages is described in [11].

Schneier and Riordan [21] present a protocol where the TTP acts as a public publishing location (which might be implemented as a secure database server). The authors describe both an on-line and an off-line version of the protocol. We note that the off-line solution requires a *visible* TTP, since the form of the receipt changes depending on whether the trusted entity was invoked or not. Moreover, the TTP must be directly involved in any secondary adjudication as it must provide, in the case involving dispute resolution, an additional signed proof-of-mailing with each query or deposit.

Recently, Ateniese et al. [4] have shown how to realize hybrid schemes that combine the strengths of both the on-line and off-line approaches to achieve efficiency while involving parties that are semi-trusted rather than fully trusted.

3 An Efficient Off-line Protocol

In this section, we describe the setting in which we operate and the certified e-mail protocol built via verifiable encryption of RSA-based digital signatures.

In the rest of the paper, we will assume that the communication is carried over private and authenticated channels.

A certified e-mail protocol should minimally provide:

- **Fairness:** No party should be able to interrupt or corrupt the protocol to force an outcome to his/her advantage. The protocol should terminate with either party having obtained the desired information, or with neither one acquiring anything useful.
- **Monotonicity:** Each exchange of information during the protocol should add validity to the final outcome. That is, the protocol should not require any messages, certificates, or signatures to be revoked to guarantee a proper termination of the protocol. This is important, because if revocation in needed to ensure fairness, then the verification of the validity of the protocol outcome becomes a bottleneck as it requires TTP's active participation.
- TTP **invisibility:** A TTP is *visible* if the end result of an exchange makes it obvious that the TTP participated during the protocol.
- **Timeliness:** It guarantees that both parties will achieve their desired items in the exchange within finite time.

Occasionally, it is desirable to keep the content of confidential e-mails secret from trusted parties acting as intermediaries. Thus, an optional feature is:

- **Confidentiality:** In case the exchange is deemed confidential, the protocol should not need to disclose the message contents to any other party excepting the sender and the recipient.

3.1 Our Setting and Verifiable Encryption of RSA Signatures

Let n be the product of two large primes p and q, such that factoring n is computationally infeasible without knowing p or q. It is generally convenient to work inside some cyclic subgroup of large order. For that reason, we generate p and q as *safe* primes, i.e., $p = 2p' + 1$ and $q = 2q' + 1$ where p' and q' are primes. We then consider the subgroup $QR(n)$ of quadratic residues, i.e., $QR(n)$ contains all the elements y such that there exists x with $y = x^2$. It is easy to see that $QR(n)$ is a cyclic group of order $p'q'$. Finding a generator is also straightforward: randomly select $\bar{g}$ and compute the generator $g = \bar{g}^2$. Since elements in $QR(n)$ have orders p', q', or $p'q'$, the order of g will be $p'q'$ with overwhelming probability.

We can also describe a proof of knowledge that allows a prover to convince a verifier of the equality of discrete logarithms. Let $g, h \in QR(n)$ be publicly known generators. The prover selects a secret x and computes $y_1 = g^x$ and $y_2 = h^x$. The prover must convince the verifier that:

$$\mathrm{Dlog}_g\, y_1 = \mathrm{Dlog}_h\, y_2.$$

The protocol, drawn from [9], is run as follows:

1. The prover randomly chooses t and sends $(a, b) = (g^t, h^t)$ to the verifier.
2. The verifier chooses a random challenge $c \in \{0,1\}^{160}$ and sends it to the prover.
3. The prover sends $s = t - cx \bmod p'q'$ to the verifier.
4. The verifier accepts the proof if:

$$a = g^s y_1^c \quad \text{and} \quad b = h^s y_2^c.$$

To turn the protocol above into a signature on an arbitrary message m, the signer can compute the pair (c, s) as:

$$c = \mathcal{H}(m \| y_1 \| y_2 \| g \| h \| g^t \| h^t), \; s = t - cx \bmod p'q'.$$

where $\mathcal{H}(\cdot)$ is a suitable hash function. To verify the signature (c, s) on m, it is sufficient to check whether $c' = c$, where

$$c' = \mathcal{H}(m \| y_1 \| y_2 \| g \| h \| g^s y_1^c \| h^s y_2^c).$$

Following the notation in [3], we will denote an instance of this signature technique by $EQ\_DLOG(m; g_1^x, g_2^x; g_1, g_2)$. Substantially, $EQ\_DLOG(\cdot)$ is a Schnorr-like signature [23] based on a proof of knowledge performed non-interactively making use of an ideal hash function $\mathcal{H}(\cdot)$ (*à la* Fiat-Shamir [14]).

In [3] it is shown that it is possible to define very efficient protocols for verifiable encryption of several digital signature schemes. Given an instance S of a digital signature on an arbitrary message, we say that $VE(S)$ is a TTP-verifiable encryption of S, if such an encryption can be verified to contain S in a way that no useful information is revealed about S itself. Only TTP is able to recover the signature from the encryption $VE(S)$.

We focus our attention on RSA signatures [22], that is, if (e, n) is a public key with e prime then the secret key is d such that $ed \equiv 1 \bmod 2p'q'$. To sign a message m, it is sufficient to compute $C = R(m)^d$, where $R(\cdot)$ is a publicly known redundancy function as defined in PKCS#1, ISO/IEC 9796, etc. (see [17] p. 442). For the sake of simplicity, we employ the hash-and-sign paradigm and assume that $R(\cdot)$ is a suitable hash function such as SHA-1. The signature is accepted only if $C^e = R(m)$. The encryption algorithm used to encrypt the RSA signature is the ElGamal algorithm: given a secret key x and a corresponding public key g^x, a message s is encrypted by generating a random r and computing $K_1 = sg^{xr}$, $K_2 = g^r$. The value s can be recovered by computing $s = K_1/(K_2)^x$.

Each user first runs a one-time *initialization phase* by which the user and the trusted third party $\mathcal{T}$ agree on common parameters. More specifically (see [3] for details):

1. The user, U say, sends (e, n) to $\mathcal{T}$;
2. $\mathcal{T}$ authenticates the user then selects a random base $\bar{g}$ and a random exponent x. It computes $g = \bar{g}^2$ and sends $CERT_{\mathcal{T}:U} = Sign_{\mathcal{T}}(g, y = g^x, U, (e, n))$ to the user, where $Sign_{\mathcal{T}}(\cdot)$ denotes a signature computed by $\mathcal{T}$.

It is assumed that the user provides a proof of n being a product of safe primes (see [8]). The signature $CERT_{\mathcal{T}:U}$ is a publicly known certificate and, in a real-world implementation, will contain relevant information including protocol headers, timestamp, transaction ID, and certificate lifetime. Notice that, the trusted party $\mathcal{T}$ does not need to store the secret exponent x for each user. In fact, such a secret can be inserted into $CERT_{\mathcal{T}:U}$ encrypted via a symmetric encryption algorithm. Thus, $\mathcal{T}$ needs to store only one value, the symmetric key, for all the users.

The computation of a verifiable encryption of a RSA signature on a message m is performed as shown in [3]. In particular, the user U encrypts via ElGamal the signature $R(m)^d$ by computing a random r and:

$$K_1 = R(m)^{2d} y^r \text{ and } K_2 = g^r.$$

Notice that, the signature $R(m)^d$ is squared to make sure that the value encrypted belongs to the set of quadratic residues $QR(n)$. Then, the user provides evidences that the encryption has been correctly computed by showing that:

$$\text{Dlog}_{y^e}(K_1^e / R(m)^2) = \text{Dlog}_g(K_2),$$

and this is done via $EQ\_DLOG(\cdot)$ w.r.t. the message m. (Observe that the verifier should recover the bases $y = g^x$ and g from $CERT_{\mathcal{T}:A}$.) We will denote the verifiable encryption of a RSA signature, $R(m)^d$, with $VE_{\mathcal{T}}(R(m)^{2d})$. Let U denote a generic user, the value $VE_{\mathcal{T}}(R(m)^{2d})$ contains: $CERT_{\mathcal{T}:U}$, the ElGamal encryption of $R(m)^{2d}$, and the signature of knowledge of the equality of discrete logarithms ($EQ\_DLOG(\cdot)$).

3.2 The Protocol

A certified e-mail protocol using verifiable encryption is shown in Figure 1. Consider a scenario in which the sender A sends a message to B and wants a receipt signed by B in exchange. The recipient B has to generate and sign the receipt before being able to read the content of the message. The protocol has to provide fairness, specifically, it must ensure that the sender receives the receipt if and only if the recipient can read the message. The protocol is designed so that the TTP is invoked only in case of dispute. As long as both A and B are honest, there is no need to involve the trusted entity in the protocol. This is a big advantage compared to on-line protocols in which a trusted entity is needed for each transaction.

Moreover, the protocol is designed to make sure that A cannot misbehave. Only B is allowed to cheat by not sending the message in the last step. This feature is highly desirable in the setting of certified e-mail, as the recipient would prefer to assume a *passive role* rather than being actively involved in dispute resolutions. Notice that, in a certified e-mail protocol, the sender initiates the exchange process, thus it is natural to desire that the recipient of the message be relieved by any burden caused by malicious senders.

User B receives the certificate $CERT_{\mathcal{T}:B}$ by engaging in an initialization phase with the trusted party $\mathcal{T}$ as explained in the previous section. Similarly, B's public key is (e, n) with e prime and n product of safe primes and $QR(n)$ is the subgroup of squares in which we operate. The protocol consists of the following steps (operations are taken modulo n):

- **Step 1** The sender A selects a random r, computes $y = r^e R(m)$, and signs it including a protocol header PH. Such a signature, denoted by S_A, is sent to B.
- **Step 2** The recipient B squares y and computes $(y^2)^d = r^2 R(m)^{2d}$. It then computes the verifiable encryption of y^{2d}, $VE_{\mathcal{T}}(y^{2d})$, and sends the result to A. However, B has to sign the result in order to include a protocol header $\overline{PH}$ and the sender's signature S_A. More importantly, B's signature (S_B) makes it possible to neutralize *malleability* attacks against the ElGamal encryption and also preserves B's *protocol view* at that specific point in time.
- **Step 3** After receiving the message from B, A verifies the signature and that the encryption contains the correct receipt. If that is the case, A sends m to B.
- **Step 4** The recipient B reads the message m and sends the receipt $Rec = R(m)^d$ to A.

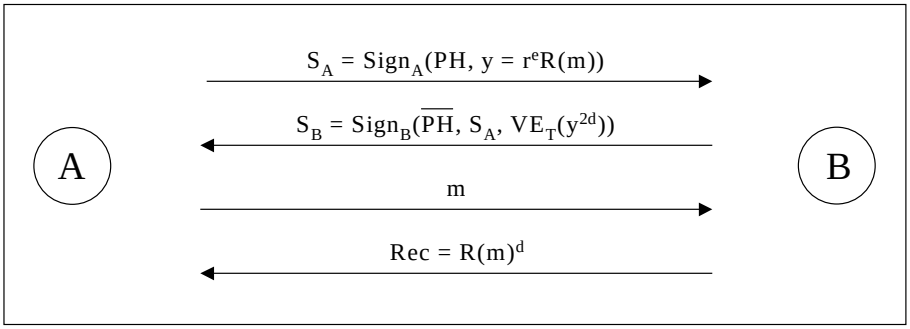

Fig. 1. Off-line certified e-mail protocol via verifiable encryption of RSA signatures

Notice that, a certified e-mail protocol is not a simultaneous exchange of items but rather a *asymmetric* exchange since the message has to be sent first to allow the recipient to compute a corresponding receipt based on the message received. This fact has some positive side effects on our scheme. For instance, the recipient does not need to include any *time limit* into the signature S_B since the decisions of sending a particular message and when this has to happen are taken exclusively by the sender.

If B does not send the receipt in Step 4, then A contacts the trusted entity and both run the following protocol:

- **Step 1** A sends B's signature, S_B, to the TTP along with r and m.
- **Step 2** The TTP verifies first the signatures S_A and S_B (S_A is contained in S_B). Then, it recovers y^{2d} from the verifiable encryption and computes y^d (see Remark 2 below). Finally, the TTP checks whether the value $s = y^d/r$ is indeed a valid signature of the message m under B's public key (i.e, it checks whether $s^e = R(m)$). If so, it sends s to A and forwards m to B.

The TTP has to forward the message m to B to nullify any attempts of the sender A to successfully retrieve a receipt without revealing the message m to B. Specifically, A may not have sent the message m in Step 3 above.

The protocol fairness is built around the assumption that the sender A can verify that the verifiable encryption indeed contains a valid receipt. Only the TTP can recover the receipt from the verifiable encryption.

Remark 1. The protocol headers, PH and $\overline{PH}$, contain relevant information such as the identities of the parties involved (A,B, and TTP), the cryptographic algorithms employed, timestamps and transaction IDs to prevent replay attacks, and other pertinent information about the protocol.

Remark 2. Notice that, the recipient B squares the value $y = r^e R(m)$ sent by A to make sure that it is signing an element of the set $QR(n)$ (d is odd since e is prime). Only the recipient B knows the factorization of n and it is usually infeasible to compute square roots modulo n without knowing the factors of n. However, given $z = y^{2d}$, the TTP can efficiently compute y^d by employing the following well-known method, based on the Euclidean algorithm, which we report here for convenience:

1. observe that $z^e = y^2$ and that $gcd(e, 2) = 1$;
2. we can use the extended Euclidean algorithm to compute two integers u, v such that $u2 = 1 + ve$ in $\mathbb{Z}$;
3. raising both terms of the equation $z^e = y^2$ by u, we obtain: $z^{ue} = y^{u2} = y^{1+ve} = yy^{ve}$
4. thus we have: $z^{ue}y^{-ve} = y$, or $(z^u y^{-v})^e = y$;
5. it is now clear that the term $z^u y^{-v}$ is congruent to y^d (modulo n).

Remark 3. In our protocol, the sender A has to reveal the message m to the TTP in case of dispute. If message privacy has to be preserved, it is sufficient to substitute m with $\overline{\overline{PH}}||P_B(m)$ in the protocol above, where $P_B(\cdot)$ represents the public-key encryption under B's public key and $\overline{\overline{PH}}$ is a protocol header. Notice that the receipt assumes a new format:

$$Rec = R(\overline{\overline{PH}}||P_B(m))^d,$$

which has to be interpreted in a special way: it is considered a valid receipt of the message m only when accompanied by m and $\overline{\overline{PH}}$ such that:

$$(Rec)^e = R(\overline{\overline{PH}}||P_B(m)).$$

The public-key encryption $P_B(\cdot)$ should be deterministic or, if randomized, the sender A must reveal the random parameters used to encrypt the message. The approach we have taken for the implementation of the protocol is to encrypt the message m as $E_k(MAC_l(m)||m)$, $P_B(k||l)$, where: k, l are random secret values; $MAC_l(\cdot)$ is a MAC function, such as HMAC-SHA-1; $P_B(\cdot)$ is a deterministic public-key encryption algorithm, such as plain RSA; $E_k(\cdot)$ is a symmetric-key encryption algorithm, such as AES in CBC mode.

The new protocol header $\overline{\overline{PH}}$ has to be checked, either by B or the TTP, to contain the correct information relevant to the protocol. Moreover, it has to clearly state that the receipt Rec has to be interpreted in the special way described above.

Remark 4. The certified e-mail protocol presented above works for RSA signatures but can easily be extended to work for other schemes based on a similar setting such as Rabin and Guillou-Quisquater [16] signature schemes, or provably unforgeable signature schemes such as Cramer-Shoup [10] and Gennaro-Halevi-Rabin [15]. In [3], it is shown how to extend the verifiable encryption protocol to work with such popular signature schemes.

4 Analysis and Comparisons

In this section we present an analysis of our protocol and we compare it with the state-of-the-art in the field. Our claim is the following:

Claim: *The protocol above is a certified e-mail protocol which provides fairness, monotonicity, timeliness, and TTP invisibility. Moreover, the protocol optionally provides confidentiality of the message, i.e., the arbitration can be performed without revealing the e-mail content to the trusted intermediary.*

Sketch: Clearly our protocol provides TTP invisibility since the structure of the receipt does not indicate whether the TTP was involved or not in dispute resolutions. The protocol provides also monotonicity since any signature (including the receipt) will not be revoked in order to guarantee a proper termination of the protocol. Confidentiality is achieved by encrypting the actual message content in such a way that only the recipient can *open* it and this is achieved through standard encryption technology.

We assume only *resilient* channels. A resilient channel will eventually deliver a message sent through it within a time lapse which may be arbitrarily long, yet finite. Moreover, the recipient does not need to include any *time limit* into the signature S_B and the sender A has the ability to decide to abort the protocol and adopt a scheme for protocol resolution that can be executed in a finite period of time. Therefore, our protocol provides timeliness.

Regarding fairness, it is sufficient to prove that: a message is read by the recipient B if and only if the sender A gets the corresponding receipt. Observe that the first two messages of our protocol are used just to collect evidences by which the sender A can solve disputes by interacting with the TTP. If the TTP is not invoked then the relevant protocol messages are only those in Step 3 and Step 4 where a message m is sent in exchange of the corresponding receipt. Therefore, fairness is preserved in this case. If the TTP is invoked (by A) then B's signature (S_B) will be sent to the TTP along with the message m and the blinding factor r. The TTP will compute y^d from the verifiable encryption and will check whether:

$$(y^d/r)^e = R(m).$$

If that is the case, then A and B receive y^d/r and m, respectively. Hence, even in this case fairness is preserved since the sender will receive a signature on a message which is forwarded to the recipient. □

It is interesting to notice that the recipient does not need to contact the TTP in case of dispute. This feature makes our protocols very attractive in real-world environments in which recipients would prefer to assume a passive role rather than being actively involved in dispute resolutions caused by malicious senders. More importantly, the recipient is *stateless* in the sense that it does not need to store state information regarding the transactions in which he is involved[1]. Indeed, the recipient may not store anything about the first two steps of the protocol and, in principle, the message embedded by the sender in the value y in Step 1 of the protocol could be different from the message sent in Step 3. Obviously, this does not violate the fairness property since the sender A cannot use the message in Step 2, since it is encrypted, unless he contacts the TTP. However, the TTP will always forward the corresponding message to B, thus neutralizing de facto any attempt of the sender to force an outcome of the protocol to his advantage.

We compare now our protocol with previously proposed protocols. Some of the off-line protocols are not monotonic, for instance, the protocol in [2] requires signatures to be revoked in order to guarantee fairness. Among monotonic and off-line protocols, we believe those in [18, 1] represent the state-of-the-art in the field. The work of Micali [18] shows that it is possible to achieve a simple certified e-mail protocol with only three messages (one less than our protocol). However, it should be noticed that:

1. the recipient of the message has to be actively involved in the dispute resolution and is forced to keep state;
2. a time limit has to be incorporated into the message by the sender to force the recipient to send the receipt within a specified period of time. This has to be done in order to guarantee fairness;
3. a *reliable* channel (as opposed to a resilient channel) is required between the recipient and the trusted third party.

[1] Notice however that implementations of the certified email scheme may require the recipient to store certain state information.

A channel is reliable when it is always operational and operates without delays. It is very difficult to build a reliable channel in some network environments, such as wireless networks. This fact may limit the applicability of the protocol in [18]. Furthermore, for each message received, the recipient is forced to communicate with the trusted intermediary in case of dispute and such a communication has to happen before the time limit expires.

The work in [1] presents a fair-exchange protocol which is provably secure in the random oracle model. The authors specialize their protocol to work as an off-line certified e-mail scheme that, similarly to our protocol, requires only resilient channels. Their protocol is based on *verifiable escrow schemes*, which essentially are verifiable encryptions where each encryption comes with an attached *condition* that specifies a decryption policy. As our scheme, their protocol works for a broad range of signature schemes. However, the scheme in [1] has some drawbacks, in particular:

1. it is expensive in terms of communication complexity, performance, and amount of data transmitted. This is mainly due to the *cut-and-choose* interactive proof technique employed to achieve a verifiable escrow.
2. the recipient has to keep state and both the sender and the recipient have to be actively involved in dispute resolutions;
3. the trusted third party needs to keep state.

Notice that both protocols [18, 1] and the version of our protocol that provides confidentiality are *invasive*, that is, the receipt generated by the receiver is not a regular signature but has to be interpreted in a special way. Our original protocol, however, is *non-invasive* as the receipt is precisely the signature of the recipient on the message received.

We believe it is important to have a *stateless* recipient who is not involved in dispute resolution protocols. Imagine a scenario in which a user receives hundreds of messages and is forced to keep track of all of them, store state information, and engage in protocol resolutions with the TTP in case of dispute. This may be very unappealing for users, in particular for those operating in environments where servers may frequently *crash*, losing state information. In some case (such as in [18]), operations have to be made before a time limit expires which may make even impossible to guarantee fairness in some environments.

Stateless-recipient protocols may be very useful when users are equipped with mobile devices such as cellular phones or wireless PDAs. Indeed, mobile devices are often switched off, which may cause some time limits to expire without giving the possibility to run any dispute resolution protocol [2].

5 System Architecture

In this section we present a description of the system we implemented. First, we give a brief overview of the technology behind electronic mail systems, then

[2] For instance, it is required to leave mobile devices off in proximity of hospitals or inside airplanes. Devices could also turn themselves off when, for instance, batteries are flat.

we show how to establish forms of interaction between our system and other standard modules involved in the process of delivering electronic mail contents and, finally, we describe in detail our system. We called our prototype implementation TURMS, an Etruscan god, messenger of the gods and guide of the deceased to the underworld.

5.1 Overview of Electronic Mail

Electronic mail is today probably one of the most used services on the Internet. It provides support to send a message to a destination. The message is passed from one computer to another, often through computer networks and/or via modems over telephone lines. The process of sending, delivering and receiving e-mail is specified in some standards and makes use of three types of programs, each of them with a specific task.

A *Mail User Agent* (MUA) is a program that allows the user to compose and read electronic mail messages. It provides the interface between the user and the Mail Transfer Agent (MTA). Outgoing mail is passed to an MTA for delivery while the incoming messages are picked up from a MTA. A MUA can also pick up mail remotely from, for instance, a POP3 server, via POP3 protocol.

A *Mail Transfer Agent* (MTA) is a system program which accepts messages from the MUA and routes them to their destinations. It sometimes delivers mail into each user's system mailbox. A MTA can also communicate with other MTA programs via the SMTP protocol, in order to deliver mail remotely. MTAs are responsible for properly routing messages to their destination, using so-called Mail eXchanger (MX) records. Mail eXchanger records are maintained by domain name servers (DNS) and tell MTAs where to send mail messages. More precisely, they tell an MTA which intermediate hosts should be used to deliver a message to the target host. The MX records vary depending on the domain.

A *Mail Delivery Agent* (MDA) is used to place a message into a user's mail-box. When the message arrives at its destination, the MTA will give the message to the appropriate MDA, who will add the message to the user's mail-box.

Our system works at the transport level, in connection with Exim [13], a mail transport agent. This approach has the advantage that allows a TURMS user to handle mail using theoretically any MUA.

5.2 TURMS and Exim

Exim [13] is a mail transfer agent developed at the University of Cambridge designed to work efficiently on systems that are permanently connected to the Internet and are handling a general mix of mail. In addition, with special configuration, Exim can act as a mail delivery agent.

Exim was built having a decentralized architecture, so there is no central process performing overall management of mail delivery, but some DBM files are maintained to make the delivery more efficient in some cases. The system implements flexible retry algorithms, used for directing and routing addresses and for delivery.

The system can handle a number of independent local domains on the same machine and provides support for multiple user mailboxes controlled by prefixes or suffixes on the user name.

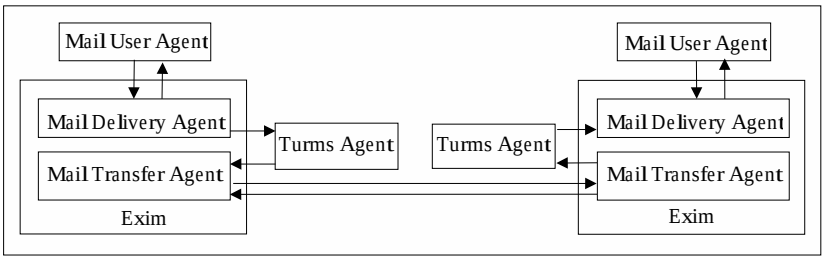

Fig. 2. TURMS and Exim communication

The main delivery processing elements (drivers) of Exim are called directors, routers, and transports.

A *director* is a driver that operates on a local address, either determining how to deliver the message, or converting the address into one or more new addresses (for example, via an alias file). A local address is one whose domain matches an entry in the list given in the 'local_domains' option, or has been determined to be local by a router. The fact that an address is local does not imply that the message has to be delivered locally; it can be directed either to a local or to a remote transport.

A *transport* is a driver that transmits a copy of the message from Exim to some destination. There are two kinds of transport: for a local transport, the destination is a file or a pipe on the local host, while for a remote transport the destination is some other host. A message is passed to a specific transport as a result of successful directing or routing. If a message has several recipients, it may be passed to a number of different transports.

A *router* is a driver that operates on an apparently remote address, that is an address whose domain does not match anything in the list given in 'local_domains'. When a router succeeds it can route an address either to a local or to a remote transport, or it can change the domain, and pass the address on to subsequent routers.

Our system takes advantage of the following features of Exim: the ability to allow a message to be piped to another program, the possibility to have multiple user mailboxes controlled by prefixes or suffixes on the user name and the ability to send messages (to remote or local e-mail addresses) using Exim.

These features allow a process, the Turms_agent, to intercept messages before they are delivered to a user and process them. Also, when needed, auxiliary messages can be generated by the TURMS agent and sent using Exim.

The message exchange mechanism between a local TURMS agent and Exim is presented in Figure 2. Special entries in Exim's configuration file indicate to

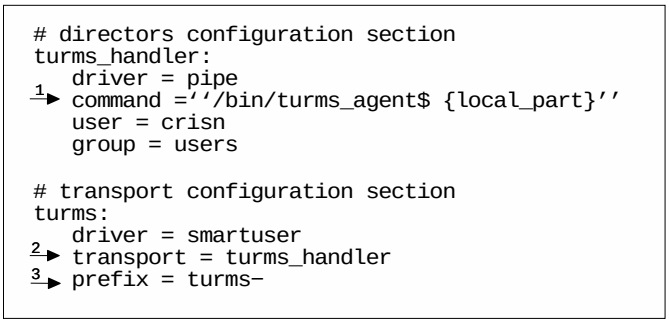

Fig. 3. Exim configuration file - TURMS entries

Exim that mail delivered to an user containing a certain prefix, *turms-* in our case (see Figure 3, Mark3), to be delivered using a specific transport, *turms_handler* in our case (see Figure 3, Mark 2). The transport specified for that prefix is a pipe to a program, Turms_agent (see Figure 3, Mark 1). This way, the message send to a local or remote user is delivered to a local Turms_agent program.

When Turms_agent receives a message, it processes it according to the protocol specifications, it generates a new message and it sends it to the real user using Exim. If the message is sent to a domain for which Exim does not do local delivery, the message will be sent via SMTP to another Exim server on another machine. There, because of the prefix mechanism, the message will be delivered to a local Turms_agent process which in turn will send a message to the local user, by using Exim. If the message is sent to a domain for which Exim does local delivery, the message will be delivered either to the end user or the Turms_agent, depending on the protocol. The delivery address specifies to Exim if the destination is a Turms_agent process or a local user inbox. Note that all the Exim servers running the certified e-mail protocol need to be configured to deliver special messages to a local TURMS agent, responsible of implementing the certified e-mail protocol.

5.3 System Implementation

We have actually implemented a slightly modified version of the protocol in Section 3.2. In particular, the value y in Step 1 is now $MAC_k(m)$, for a suitable MAC function such as HMAC-SHA-1. In Step 3, the sender reveals m and also k and the recipient checks whether the message m is the same received in Step 1. The receipt is a signature on the MAC function.

The protocol itself requires an initialization phase which has to be done only once, and in case of dispute, a recovery phase. We provide support for all of these operations. The system consists of a web-based interface, Turms_CA, providing support for the initialization and recovery off-line phases, and of an MTA, Turms_agent, implementing the certified e-mail protocol.

One of the design features we considered was encapsulating all the cryptographic operations in a library, used by both Turms_CA and Turms_agent. The library makes use of the openssl [19] library and provides facilities such as: strong RSA keys generation and managing, TURMS certificate definition and management (generating, signing, verifying), operations on Schnorr-like objects used in our protocol, RSA verifiable encryption definition and management (generation, verifying, ability to extract the RSA signature out of the verifiable encryption). In addition, the library also provides conversion from the computation data format (openssl specific data structure) to communication format for both verifiable encryption and TURMS certificate entities. For each of these data structures, the library provides a simple and easy to use API.

Turms_CA provides an interface allowing users to register and to obtain a certificate. The user submits his public RSA key along with some additional information and will receive the corresponding TURMS certificate In addition, for each user, we have decided to save the corresponding secret of the CA rather than incorporating it into the certificate This secret is used to extract RSA signatures out of a verifiable encryptions.

Turms_CA can also solve disputes. A user can submit a claim including the verifiable encryption file and the message. The CA computes the signature out of the verifiable encryption information, then sends the signature to the user that submitted the claim and the message to the user whose verifiable encryption was submitted.

The core of the system is the *Turms_agent* program which implements the certified e-mail protocol. *Turms_agent* is a stateless program, a different instance of the program is invoked with every new message. The state of the protocol for different messages is saved on the disk. Every state of the protocol for a message has a correspondent file saved on disk. A message is uniquely identified by a concatenation of the process id, host id and current time. Every state also has a unique identifier associated with it. The security of the channel between two Turms_agents is achieved via Blowfish encryption with a size key of 16 bytes. We used an HMAC with a key size of 10 bytes to obtain a randomized one-way function of the message. Turms_agent also logs for each transaction information about the main important steps.

The protocol consists of a sequence of actions taken by a Turms_agent program upon receiving a message. Every message is associated with a specific transaction. The type of the message along with the transaction identifier are specified in the destination address. A transaction is opened when a user attempts to send a certified e-mail message and it ends in one of the following cases: the exchange was performed correctly, the exchange was canceled, or the exchange started but the recipient did not send the receipt. The protocol uses the following types of messages:

- Original_Mess is the message that has to be sent. It is generated by a MUA, no transaction identifier or type is associated with it.
- Hmac_Mess contains the value of an HMAC function applied on the body of an Original_Mess. It is generated by a *Turms_agent*.

- Invitation_Mess is the message that notifies a user about a certified e-mail message. It is generated by a $Turms\_agent$.
- Cancel_Mess is an Invitation_Mess which has the Subject field consisting only of the word 'Cancel' indicating that the transaction was refused. The body of the message is ignored. It is generated by a MUA, in reply to an Invitation_Mess coming from a $Turms\_agent$.
- Accept_Mess message is similar as structure with the Cancel_Mess, but the Subject field consists of the word 'Accept' indicating that the transaction was accepted. The body of the message is ignored. It is generated by a MUA, in reply to an Invitation_Mess coming from a $Turms\_agent$.
- Transaction_Canceled_Mess indicates that a transaction was canceled. It is generated by a $Turms\_agent$.
- Verifiable_Encryption_Mess contains a verifiable encryption message. It is generated by a $Turms\_agent$.
- Original_Mess_and_Key contains the body of an Original_Mess and the key used to compute the HMAC value applied on the Original_Mess that was sent in the corresponding (has the same transaction identifier) Hmac_Mess. It is generated by a $Turms\_agent$.
- Signature_Mess contains a RSA signature of an HMAC value of an Original_Mess message. It is generated by a $Turms\_agent$.

In response to an event, a $Turms\_agent$ can take actions that will result in sending messages and/or saving additional data on the disk. Consider a scenario in which the sender A wants to send a certified e-mail to B. We make use of the following notation: A's domain is denoted by $domain_A$ and B's domain is denoted by $domain_B$, the $Turms\_agent$ programs corresponding to A's mail server and B's mail server are denoted by $turms\_agent_A$ and $turms\_agent_B$, respectively . Finally, $user_A$ and $user_B$ represent the MUAs at A's site and B's site, respectively.
The protocol consists of the following steps:
Step 1. $user_A$ sends the Original_Mess message. $user_A$ sends the Original_Mess message to an alias defined for B, $certified\_B$ say, which includes the address to which the mail is sent. The address should be prefixed with the prefix specified in the Exim configuration file (see Figure 3), should be sent to the local mail server and should contain enough information to allow to recover the actual remote address. Exim will deliver the message via the pipe transport to the $turms\_agent_A$ program. The result of this step is that $turms\_agent_A$ will receive the Original_Mess.
Step 2. $turms\_agent_A$ receives the Original_Mess message. When the message is received by $turms\_agent_A$, the agent creates a unique identifier for this new transaction: $timestamp - process\_id - host\_id$. Then it processes the message, saves the body of the message along with some header information. It generates a key that will be used to compute a HMAC of the body of the message, saves the key on the disk, computes HMAC of the message, saves the content of the message, recovers the real address, generates a Hmac_Mess with the HMAC value just computed and sends it to $turms\_agent_B$. The type of the message along with

the identifier is specified in the address. Also the reply address is updated such that the mail appears as coming from a *turms_agent* (by adding the prefix). The result of this step is that $turms\_agent_B$ will receive a Hmac_Mess.

Step 3. $turms\_agent_B$ receives the Hmac_Mess message. When $turms\_agent_B$ receives the Hmac_Mess, it saves this information on the disk and generates an Invitation_Mess, and sends it to $user_B$, notifying about a certified e-mail message for him, and asking him to reply to this message with 'Accept' written in the subject, if he accepts the message, or 'Cancel' if he refuses it. The result of this step is that $user_B$ will receive an Invitation_Mess.

Step 4. $user_B$ receives the Invitation_Mess message. When $user_B$ receives the Invitation_Mess, he will reply either with 'Accept' or 'Cancel' in the subject. The result of this step is that $turms\_agent_B$ will receive either an Accept_Mess or a Cancel_Mess.

Step 5. $turms\_agent_B$ receives the Cancel_Mess message. When $turms\_agent_B$ receives the Cancel_Mess, it generates and sends two Transaction_Canceled-_Mess messages, one to $user_B$ and the other to $turms\_agent_A$. The transaction is closed. The result of this step is that both $user_B$ and $turms\_user_A$ will receive a Transaction_Canceled_Mess.

Step 6. $turms\_agent_B$ receives the Accept_Mess message. When $turms\_agent_B$ receives the Accept_Mess, it recovers the HMAC value of the message from the disk, then computes the verifiable encryption of B's signature, and generates a Verifiable_Encryption_Mess message and sends it to $turms\_agent_A$. The result of this step is that $turms\_agent_A$ will receive a Verifiable_Encryption_Mess.

Step 7. $turms\_agent_B$ receives the Transaction_Canceled_Mess message. When $turms\_agent_B$ receives the Transaction_Canceled_Mess, he sends it to $user_A$ and deletes the HMAC and the OriginalL_Mess information saved on the disk in step 2. The result of this step is that $user_A$ will receive a Transaction_Canceled_Mess.

Step 8. $turms\_agent_A$ receives the Verifiable_Encryption_Mess message. When $turms\_agent_A$ receives the Verifiable_Encryption_Mess, it verifies that the message indeed contains a RSA signature. If yes, it recovers from the disk both the body of the Original_Mess and the key used to compute HMAC value, generates the Original_Mess_and_Key message by concatenating the key to the message and then sends it to $turms\_agent_B$. Also the Verifiable_Encryption_Mess is sent to $user_A$. The result of this step is that $turms\_agent_B$ will receive an Original_Mess_and_Key message and $user_A$ will receive a Verifiable_Encryption_Mess.

Step 9. $turms\_agent_B$ receives an Original_Mess_and_Key message. Upon receiving an Original_Mess_and_Key message, $turms-\_agent_B$ computes a HMAC on the body the received message with the key he just received and it compares it with the HMAC data saved on the disk in Step 2. If they are the same, it computes B's RSA signature on the HMAC, generates a Signature_Mess message and sends it to $turms\_agent_A$. If the two HMAC values are not the same, then $turms\_agent_B$ generates and sends two Transaction_Canceled_Mess messages, one to $user_B$ and the other to $turms\_agent_A$. The result of this step consists of two messages: $user_B$ receives the Original_Mess and $turms\_agent_A$

receives a Signature_Mess, or both $user_B$ and $turms\_agent_A$ receive a Transaction_Canceled_Mess.

Step 10. $turms\_agent_A$ receives a Signature_Mess message. When $turms\_agent_A$ receives Signature_Mess, it sends it to $user_A$ and cleans all the auxiliary files used during the transaction.

In order to have the protocol described above working correctly in the case when the message contains one or more attached files, some additional processing needs to be done: the corresponding MIME information from the message header needs to be saved when the original message is processed. This information is used when the message is finally sent to the user (step 7), to make sure that MUA understands that the message carries some attached files.

We used the following set up for developing and testing our system. The web interface running on a Linux machine dual 450MHz Pentium II, 128MB RAM, running Apache Web Server. We tested our system in a configuration of two virtual domain names (securemail1.cs.jhu.edu and securemail2.cs.jhu.edu), each served by an Exim server version 3.14. The machines were 300MHz Pentium II, 256 Mb RAM, Linux boxes (2.16 kernel). The program PINE [20] was used as Mail User Agent.

6 Conclusions

We presented a very efficient off-line certified e-mail system. Both the recipient and the TTP can be set to be stateless and the recipient can assume a passive role without being involved in dispute resolutions so that the burden of solving a dispute is given only to the sender, the initiator of the protocol.

We implemented a prototype (TURMS) of the protocol and we reviewed some of the technology available today that could be employed to effectively build any certified e-mail system.

Acknowledgements

We would like to thank Theo Schlossnagle for his helpful suggestions on setting up the testing environment. Many thanks to the anonymous referees for their insightful comments.

References

1. N. Asokan, V. Shoup, and M. Waidner, "Optimistic fair exchange of digital signatures," *IEEE Journal on Selected Area in Communications*, 2000.
2. N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *Proceedings of the IEEE Symposium on Research in Security and Privacy* (I. C. S. Press, ed.), pp. 86–99, May 1998.
3. G. Ateniese, "Efficient verifiable encryption (and fair exchange) of digital signatures," in *Proceedings of the 6th ACM Conference on Computer and Communications Security*, ACM Press, 1999.

4. G. Ateniese, B. de Medeiros, M.T.Goodrich. TRICERT: Distributed Certified E-mail Schemes. In ISOC 2001 Network and Distributed System Security Symposium (NDSS'01), San Diego, CA, USA, 2001.
5. M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest, "A fair protocol for signing contracts," *IEEE Transactions on Information Theory IT-36(1)*, pp. 40–46, 1990.
6. A. Bahreman and J. D. Tygar, "Certified electronic mail," in *Proceedings of Symposium on Network and Distributed Systems Security* (I. Society, ed.), pp. 3–19, February 1994.
7. F. Bao, R. H. Deng, and W. Mao. Efficient and Practical Fair Exchange Protocols with Off-line TTP. In *IEEE Symposium on Security and Privacy*, Oakland, California, 1998.
8. J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *Advances in Cryptology – EUROCRYPT '99*, *Lecture Notes in Computer Science*, Springer-Verlag, 1999.
9. D. Chaum and T. Pedersen. Wallet databases with observers. In *Advances in Cryptology – Crypto '92*, pages 89-105, 1992.
10. R. Cramer and V. Shoup. Signature Schemes Based on the Strong RSA Assumption. In *6th ACM Conference on Computer and Communication Security*, ACM Press, 1999.
11. R. H. Deng, L. Gong, A. Lazar, and W. Wang, "Practical protocols for certified electronic e-mail," *Journal of Networks and Systems Management*, vol. 4, no. 3, pp. 279–297, 1996.
12. S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Comm. ACM 28*, no. 6, pp. 637–647, 1985.
13. "http://www.exim.org."
14. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Springer-Verlag, 1987.
15. R. Gennaro, S. Halevi, and T. Rabin. Secure signatures, without trees or random oracles. In *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139, Springer-Verlag, 1999.
16. L. C. Guillou and J. J. Quisquater. A paradoxical identity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology – CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231, Springer-Verlag, 1988.
17. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1996. ISBN 0-8493-8523-7.
18. S. Micali. Simultaneous electronic transactions. Technical Report 566420, http://www.delphion.com/cgi-bin/viewpat.cmd/US566420__, 1997.
19. OpenSSL Project team, "Openssl," May 1999. http://www.openssl.org/.
20. Pine Information Center, http://www.washington.edu/pine/.
21. J. Riordan and B. Schneier, "A certified e-mail protocol," in *13th Annual Computer Security Applications Conference*, pp. 100–106, December 1998.
22. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
23. C.P. Schnorr. Efficient signature generation by smart-cards. *Journal of Cryptology*, 4(3):161–174, 1991.
24. J. Zhou and D. Gollmann, "Certified electronic mail," in *Proceedings of Computer Security - ESORICS'96* (S. Verlag, ed.), pp. 55–61, 1996.