Certified Email with a Light On-line Trusted Third Party: Design and Implementation

Martín Abadi*
Computer Science Department
UC Santa Cruz

Neal Glew*

Bill Horne†
STAR Lab
InterTrust

Benny Pinkas
STAR Lab
InterTrust

ABSTRACT

This paper presents a new protocol for certified email. The protocol aims to combine security, scalability, easy implementation, and viable deployment. The protocol relies on a light on-line trusted third party; it can be implemented without any special software for the receiver beyond a standard email reader and web browser, and does not require any public-key infrastructure.

1. INTRODUCTION

Several protocols for certified email were developed recently and several systems for certified email are being deployed commercially (see references below). In general, their main goal is to guarantee that the receipt of an email message produces a receipt certificate whether the receiver is honest and diligent or not. They sometimes have secondary goals, such as authenticity of sender and receiver, and message confidentiality.

In order to achieve their goals, the protocols and systems often require some new assumptions and new software for email senders and receivers. Most of them also rely on some new infrastructure, and in particular on a trusted third party of some sort that serves as a mediator. There are also protocols that do not use a trusted third party and operate by having the parties do a bit-by-bit exchange of each message against the corresponding receipt.

Despite their common objectives and their common tools, theoretical and practical work on certified email tend to adopt different approaches.

In research papers, there is often an emphasis on reducing the role and the expense of a trusted third party. This emphasis has led to clever protocols with mostly off-line trusted third parties of limited power. Unfortunately, protocols that use weak trusted third parties can be hard to deploy. In particular, if the trusted third party is mostly off-line, then the sender of a message may need to be on-line for some long period after sending the message, and may need to run special-purpose server code to respond to queries by the receiver. These protocols also incur a considerable overhead on the parties that send and receive email messages.

In practice, a strong, on-line trusted third party is often adopted. This is not necessarily problematic for all involved, for it might provide good support for the business models of the companies that sell certified email. Moreover, a strong, on-line trusted third party can reduce the need for new assumptions and new software for other parties. On the other hand, on-line trusted third parties present scalability and security problems. Their overhead is typically linear in the size of the messages being emailed. Secondarily, they sometimes have access to the cleartext of all messages.

This paper presents a new protocol for certified email that aims to combine the strengths of these two approaches while addressing their shortcomings. Our approach uses an on-line trusted third party that essentially acts as a web-based escrow server (specifically, an escrow server for the exchange of an email message against a receipt for that message). The trusted third party is designed so that it does work proportional to the number of messages exchanged rather than the number of bytes exchanged, and stores no per-message state. Both of these properties contribute to the scalability of our solution. Furthermore, our protocol integrates well with the existing email and web infrastructure and is easy to implement and deploy, in the following ways.

- Our protocol is designed so that no new software is necessary for receiving a message; therefore, the receiver of a message need not take any special action in advance.[1]
- In addition, our protocol does not require the sender to run an on-line server. (Such a server is needed if a protocol requires several rounds of interaction between the sender and receiver.) Our protocol enables sending messages in a send-and-forget manner.
- The trusted third party itself is fairly simple. In particular, it does not keep any per-message state. The communication and computation overhead of the trusted third party do not depend on the size of the email messages, but rather on the number of messages.
- An implementation can primarily use standard web software, such as Java-enabled browsers and SSL. We describe an implementation of our protocol that uses such components.

*Part of this work was done when Martín and Neal were at InterTrust.

†Contact email: bhorne@intertrust.com.

[1] However, the receiver may later load special software with added features, perhaps including the sending software. Thus, the software can spread through a group of users as they receive certified email and decide to download the software and send certified emails to others.

WWW2002, May 7–11, 2002, Honolulu, Hawaii, USA.
ACM 1-58113-449-5/02/0005.

- Finally, our protocol supports several practical methods for authenticating the intended receiver of a message. We recognize that the receiver probably does not have a public-key pair, and do not propose to deploy a public-key infrastructure (PKI). We accommodate the common situations in which the receiver has a shared secret with the sender or with the trusted third party.

Comparing our protocol to previous suggestions for certified email protocols (described in Section 1.1), we believe that our protocol is particularly suitable for the existing web and email infrastructure.

The rest of this introduction reviews some related work. The following section, Section 2, further explains our assumptions and goals. Section 3 presents our basic protocol, along with an informal security analysis. Section 4 describes our implementation.

1.1 Related Work

The related work can be divided into two categories: research projects and commercial systems. We first describe relevant research projects and then review some related commercial systems.

1.1.1 *Research Projects*

Research projects can be classified according to their use of the trusted third party. Some projects use an on-line trusted third party, some use an "optimistic" approach that only requires the trusted third party to intervene in case of a dispute, and some do not involve a trusted third party at all.

Several research projects designed protocols for certified email that use a trusted third party that must be on-line during the exchange of email messages. Bahreman and Tygar [6] suggest a protocol which requires the exchange of six messages. Zhou and Gollman [27] describe a protocol which delivers the message through a trusted third party, or a sequence of trusted parties. The receiver should then sign a receipt which is routed back by the trusted parties to the server. Deng et al. [15] describe more efficient protocols that have only four messages (and are optimal in this respect). We note that all of these protocols suffer from the following drawbacks: The entire message is sent to the trusted third party (whereas in our protocol the work of the trusted third party does not depend on the size of the message). The trusted third party learns the message (which in our protocol remains hidden from the trusted third party). The protocols require the receiver, and sometimes the sender, to sign messages, implying that a PKI must be deployed. (The only requirement regarding public keys in our protocol is that the sender knows the public key of the trusted third party.)

Optimistic protocols use a trusted third party that is off-line during the actual exchange of messages. The trusted third party should operate only if there is a dispute between the parties. The protocols usually have the parties first exchange messages encrypted with the trusted third party's public key, and then exchange the messages themselves. If one of the parties defaults then the other party can contact the trusted third party and ask it to decrypt the message that was received in the first round. The main challenge is to verify, without involving the trusted third party, that the messages that are exchanged in the first round are indeed the required messages encrypted under the trusted third party's public key. The resulting protocols exchange several messages and perform several exponentiations, and therefore seem less efficient and more complicated to deploy than our protocol. The initial work in the area was done by Asokan et al. [1, 2]. Pfitzmann et al. [21] present another optimistic protocol; their main emphasis is on finding the correct definitions for the problem. Franklin and Reiter [18] describe a protocol with a semi-trusted third party that might misbehave on its own but would not collude with any of the other parties. The Tricert system [3] is a hybrid system that distributes the task of the trusted third party to less trusted postal agents that perform the on-line work, and to a trusted third party that is only involved in disputes. Ateniese and Nita-Rotaru [4] describe a design and a prototype implementation of a certified email protocol with a mostly off-line TTP. The protocol ensures that if the sender does not follow the protocol then it does not get a receipt, so the receiver never has to contact the TTP. If the receiver misbehaves then the sender can obtain a receipt by contacting the TTP. The protocol consists of four rounds, and requires changes in the email delivery systems of both sender and receiver.

Protocols that do not involve a trusted third party date back to Blum's work [7], which uses oblivious transfer, and the work of Even et al. [17], which introduced a generic method in which the parties release one bit at a time of the items they wish to exchange. (In most of the papers these items are signatures, but in our case these items are the email message and the receipt.) Improvements were suggested by other researchers [12, 14, 20]. All these protocols are fair in the sense that if one of the parties defaults then it does not have a considerable advantage over the other party, since it knows at most one more bit than the other party. (The recent work of Boneh and Naor [9] provides better fairness since it ensures that neither party can parallelize its effort to compute the bits that are unknown to it.) The main drawback of these protocols is the large number of rounds that they require (approximately as many rounds as the length of a security parameter, namely the length of keys that encrypt the messages).

1.1.2 *Commercial Systems*

There is a considerable amount of commercial work on secure and certified email. Most of this work is undocumented, and the underlying architecture of most commercial systems is not publicly available. Given the lack of information it is hard to provide a comparison with that work. In fact, one of the values of this paper may be in openly explaining one solution to the certified email problem.

RFC2298 is a standard for Message Disposition Notifications [23]. It should run across platforms and be supported by mail clients. Microsoft Outlook and Exchange server have their own return receipt mechanism. These solutions have several drawbacks: They depend on the security and tamper resistance of the email client. A corrupt receiver might break into the email client, use a filter that changes the notification fields before the message arrives at the email client, or block the return message that includes the receipt. Corporate firewalls often do this filtering process, blocking return receipts from being sent from within their corporate network. Finally, there is a problem with fair notification. If a user of Microsoft Outlook wishes to know that reading a message will generate and send a return receipt, she must either change the default settings or perform a complicated operation, involving several mouse clicks, before reading every message.

Several companies offer certified email services including Authentica [5], Certifiedmail.com [11], Zixit [28], Postx [22], ZeroGravity [26], and Sigaba [25]. Their solutions use a combination of a trusted server and a secure client. Not all of the technical details are publicly available. We note that one approach is to store the message as a web page on the trusted server and send an email to the receiver, asking it to go a URL to retrieve the message. We also note that Authentica, as in our protocol, the trusted third party stores a decryption key rather than the entire message. We do not, however, have other substantial information about this solution.

2. ASSUMPTIONS AND REQUIREMENTS

The desired protocol should allow a sender, S, to send an email message to a receiver, R, so that R reads the message if and only if S receives the corresponding return receipt. The main emphasis of our work is on ensuring the atomicity of this operation, namely ensuring that either both S and R receive their corresponding outputs, or neither of them does. Other properties of the protocol, for example the confidentiality and authenticity of the messages, are also discussed but are not the focal point of this work.

The protocol is allowed to use an on-line trusted third party, TTP. All three parties are capable of basic cryptography (both public and symmetric encryption and decryption, secure hashing, signature generation and verification). To simplify the exposition of the protocol we assume that the channels between TTP and the other parties are reliable. This property can be achieved using the communication protocol, or using specific methods that were suggested for certified email delivery by Deng et al. [15]. Furthermore, we assume that R and TTP have a secure channel between them that provides secrecy and possibly authentication of TTP to R. (This is needed when R gives secrets to TTP in order to authenticate itself.) In practice such a channel might be an SSL connection or, more generally, a channel authenticated and encrypted using symmetric keys established via a suitable protocol (e.g., Diffie-Hellman), perhaps relying on public keys.

While we assume that TTP has public keys and that these keys are known to both S and R, we do not assume that either S or R have public keys or that there is any public-key infrastructure (PKI). This means that there is an authentication problem: if TTP receives a request from a party claiming to be R and claiming it is about to read the message, how does TTP know this is really R? We assume either or both of the following authentication mechanisms: a shared secret between R and TTP and a shared secret between R and S. The former occurs when R has an account with TTP and the shared secret is the PIN number or password for the account. This mechanism allows TTP to certify that R received the email, since TTP is able to authenticate R independently of S or any other party. The latter mechanism might be based on something known through a personal or business relationship between S and R. It does not allow TTP to assert that R received the email, but only that the email was received by someone S authorized—note that S could impersonate R under this mechanism.

Turning to a more precise definition of the security requirements, the natural definition seems to be a comparison to the *ideal model*. This approach for defining security is a common strategy in many papers on cryptographic protocols. The security definition usually defines a protocol that operates in an augmented scenario, one with an additional player—the *trusted party*. Typically the protocol operates in the following way:

- The original parties send their respective inputs to the trusted party.
- The trusted party observes the inputs and computes the function that is defined by the original protocol.
- The trusted party sends to each of the other parties the respective output of the function that it should receive.

When applied to the certified email context, the ideal model protocol operates in the following way: The trusted party receives the email message from the sender, as well as a cleartext message that invites the receiver to read the message. The trusted party delivers this invitation to the receiver and obtains from it a request either to read or not to read the email. If the request is to read the email, then the trusted party sends the email to the receiver and a return receipt to the sender, otherwise the request is denied.

A protocol is defined to be secure if any information obtainable from the actual protocol could also have been obtained in the ideal model. In our scenario this corresponds to limiting the possible output of the protocol to one of two possibilities: either the receiver obtains the message and the sender obtains the return receipt, or neither of them receive the corresponding outputs.

The protocol that we suggest uses a trusted third party, TTP, and therefore it seems that it can trivially satisfy the above requirements. There are, however, several subtle issues that must be considered.

First, as discussed above, TTP must authenticate R as the receiver of the message. This is a rather complicated issue since we do not assume the existence of a PKI.

Second, we would like to hide the contents of the message from TTP. Although we assume TTP to be honest, in the worst scenario it might be curious or corrupt, and eavesdrop on the communication between S and R. If there is no shared secret between S and R then TTP can learn the message sent to R. Our protocol can, however, prevent TTP from learning the message as long as it does not eavesdrop on the communication between S and R. It can also be amended to use a shared secret between the two parties S and R to encrypt the message.

Third, we desire an efficient implementation for the protocol. In particular, the protocol in the ideal model requires the trusted party to keep state information, whereas we want to implement a stateless TTP.

3. THE BASIC PROTOCOL

This section describes our protocol for certified email using an on-line trusted third party. We start with some preliminaries, then describe the protocol, and conclude with a security analysis.

3.1 Preliminaries

3.1.1 Algorithms

The following notation is used for various cryptographic primitives.

- E(k,m) denotes an encryption of m using key k under some symmetric encryption algorithm, for example, AES in CBC mode with random initialization vectors. (We believe that integrity protection is not necessary for our protocol.)

- H(m) denotes the hash of m in some collision resistant hashing scheme, for example SHA.

- A(k,m) denotes an encryption of m using key k under some public-key encryption algorithm. The encryption algorithm should provide non-malleability (also known as security against adaptive chosen ciphertext attacks, see [16, 13]). This property means that given an encryption e it is impossible to generate a different encryption e' such that the plaintexts of both encryptions satisfy any predetermined relation.

  We suggest using for the encryption algorithm the RSA PKCS standard (using OAEP or one of its variants [24, 19, 8]), for which these properties can be proved in the random oracle model, or the Cramer-Shoup cryptosystem [13] for which these properties can be proved without any random oracle assumptions.

- S(k,m) denotes a signature of m using key k under a public-key signature algorithm, for example, the appropriate RSA standard.

- Finally $\mathsf{m}_1 \mid \cdots \mid \mathsf{m}_n$ denotes the unambiguous concatenation of the m_is, for example, using a distinguished separator or length indicators.

3.1.2 Keys and Authentication

TTP has a public key TTPEncKey that can be used to encrypt messages to TTP, and a corresponding secret key TTPDecKey. TTP also has a secret key TTPSigKey that it can use to sign messages and a public key TTPVerKey that other parties can use to verify these signatures. Only the sender S needs to know the public keys of TTP. We do not assume that the receiver R has any knowledge about these keys.

Our protocol supports four options for authenticating R: BothAuth, TTPAuth, SAuth, and NoAuth. These correspond to the combinations of authenticating R, namely TTP authenticating R or not, and S authenticating R or not. To simplify the notation we assume that TTP authenticates R using a shared secret RPwd—a password that identifies R to TTP. (In practice TTP might use a different authentication method; for example if both TTP and R reside on the same secure local-area network TTP can use R's network address to authenticate it.) The method in which S authenticates R is a query/response mechanism. R is given a query q by the receiver software and r is the response that S expects R to give. For this method of authentication to work, both S and R must establish the relationship between q and r before using the protocol, or know each other well enough to guess what the other intended. The sender picks the authentication option for each email sent, and the choice is denoted by authoption. Of course, if the sender picks authentication by TTP then TTP should be able to authenticate R; and if the sender picks authentication by S then S itself should be able to authenticate R.

3.2 The Protocol

Figure 1 shows a high-level sketch of the protocol. Let us first describe the steps of the protocol without going into detail. In step 1, the sender encrypts the message using a freshly generated key, encrypts the key with the public key of TTP, and sends this along with the encrypted message to R. In step 2, R sends the encrypted key to TTP as a request to release the key. If TTP is satisfied that the request is legitimate it sends the key to R in step 3 and sends a return receipt to S in step 4. R can then decrypt and read the message.

The protocol binds S and R to a single message in the following way: In step 1, S computes the hash of the encrypted message, encrypts it with TTP's public key, and sends this information to R. In step 2, R computes a hash of the encrypted message it receives from S, and it sends to TTP this hash and the encrypted hash that S computed. TTP compares these two values and if they are the same it concludes that S and R are obligated to the same message and continues the protocol.

In more detail, the protocol runs as follows. (Figure 2 shows this detail.)

Step 1: S performs the following operations:

1. First, S generates a random key k appropriate for use with E. S also picks authoption. If authoption is BothAuth or SAuth then S knows or generates a query q to which R should produce the response r. If authoption is TTPAuth or NoAuth then q and r should be null. It is important for S to remember m, k, q, and r for as long as the receipt may be of value.

2. S encrypts the message m with k, computing $\mathsf{em} = \mathsf{E}(\mathsf{k}, \mathsf{m})$.

3. S then computes a hash that will both identify the message to TTP and authenticate R to S. Specifically S computes $\mathsf{h}_\mathsf{S} = \mathsf{H}(\mathsf{cleartext} \mid \mathsf{q} \mid \mathsf{r} \mid \mathsf{em})$. The part cleartext is a header that asks R to read the certified email, perhaps explaining what is in the message.

4. Last, S forms a message that will ultimately be delivered to TTP by R. This message has the sender's identity (so that TTP knows where to send the receipt), the key, the authentication information, and the identifying hash: $\mathsf{S2TTP} = \mathsf{A}(\mathsf{TTPEncKey}, \mathsf{S} \mid \mathsf{authoption} \mid$ "give k to R for h_S").

5. S sends to R:

   MESSAGE 1, S to R:
   TTP | em | authoption | cleartext | q | S2TTP

(In our description of messages received during the protocol we refer to message components such as em′ and q′ rather than em and q because there is no a priori guarantee that the receiver of the message receives the original components unmodified, since an attacker may tamper with the message or may send a bogus message.)

Step 2: R, upon receipt of a message of the form $\mathsf{TTP} \mid \mathsf{em}' \mid \mathsf{authoption}' \mid \mathsf{cleartext}' \mid \mathsf{q}' \mid \mathsf{S2TTP}'$, performs the following operations:

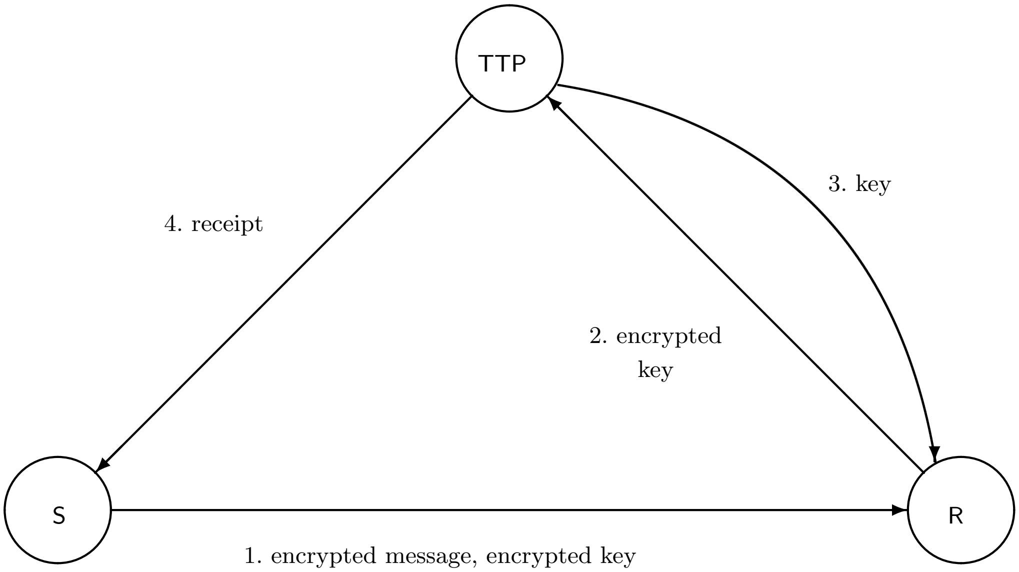

Figure 1: Protocol sketch

$$\begin{array}{lcl}
\mathsf{em} & = & \mathsf{E}(\mathsf{k}, \mathsf{m}) \\
\mathsf{h_S} & = & \mathsf{H}(\mathsf{cleartext} \mid \mathsf{q} \mid \mathsf{r} \mid \mathsf{em}) \\
\mathsf{h_R} & = & \mathsf{H}(\mathsf{cleartext}' \mid \mathsf{q}' \mid \mathsf{r}' \mid \mathsf{em}') \\
\mathsf{S2TTP} & = & \mathsf{A}(\mathsf{TTPEncKey}, \mathsf{S} \mid \mathsf{authoption} \mid \text{“give } \mathsf{k} \text{ to } \mathsf{R} \text{ for } \mathsf{h_S}\text{”})
\end{array}$$

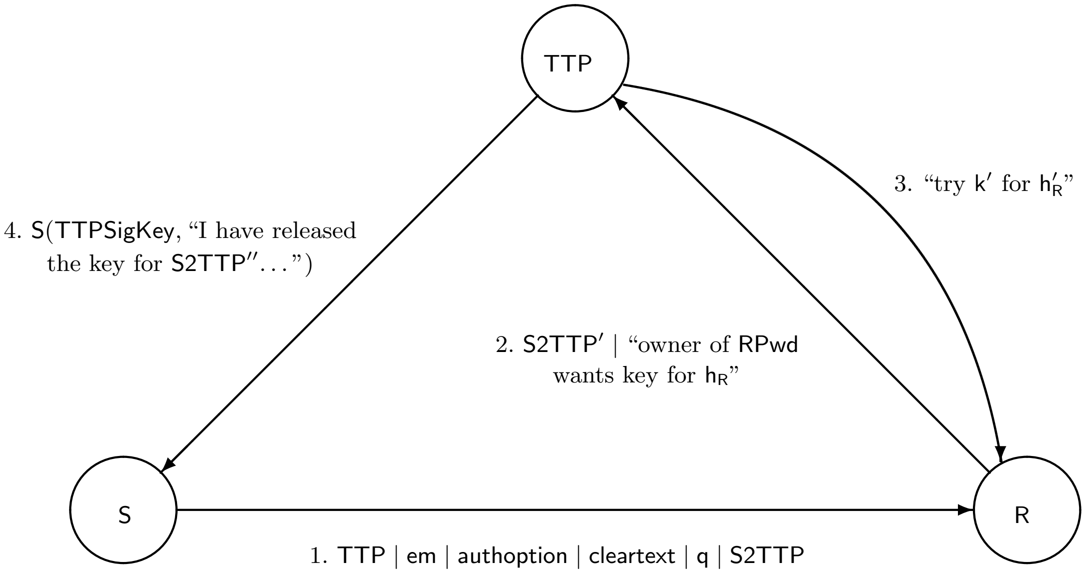
TTP
3. “try k' for h'_R”
4. $\mathsf{S}(\mathsf{TTPSigKey}$, “I have released the key for $\mathsf{S2TTP}''$. . . ”)
2. $\mathsf{S2TTP}' \mid$ “owner of RPwd wants key for $\mathsf{h_R}$”
S
R
1. $\mathsf{TTP} \mid \mathsf{em} \mid \mathsf{authoption} \mid \mathsf{cleartext} \mid \mathsf{q} \mid \mathsf{S2TTP}$

Figure 2: Protocol sketch, in more detail

1. It reads cleartext′ and decides whether it wants to read the message and is willing and able to use TTP as a trusted third party. If it does, R checks what authentication is needed. If authoption′ is SAuth or BothAuth it constructs a response r′ to query q′, while if authoption′ is TTPAuth or NoAuth then R uses null for r′. As for authenticating itself to TTP, if authoption′ is TTPAuth or BothAuth then R recalls its password for TTP, RPwd, while if authoption′ is SAuth or NoAuth then R uses null for RPwd.

2. R computes its own hash to identify the message and authenticate itself to S, h_R = H(cleartext′ | q′ | r′ | em′).

3. R sends to TTP:

   Message 2, R to TTP:
   S2TTP′ | "owner of RPwd wants key for h_R"

   This and the next message travel on the secure channel between R and TTP.

Steps 3 and 4: TTP, upon receipt of a message of the form S2TTP″ | "owner of RPwd′ wants key for h'_R", performs the following operations:

1. It tries to decrypt S2TTP″ using TTPDecKey. The plain text should be of the form S | authoption″ | "give k′ to R′ for h'_S".

2. TTP authenticates R: It checks that authoption is either SAuth or NoAuth or that RPwd′ is the password for R′. TTP then checks that h'_S equals h'_R. (This check can be regarded as the most important test done by TTP, since it verifies that S and R agree on the same message and on the query/response pair.) If TTP is unable to decrypt S2TTP″, if the plain text has the wrong form, or if any of the checks fails then TTP sends "request denied" to R and stops. Otherwise it proceeds with Messages 3 and 4.

3. TTP sends the following message to R

   Message 3, TTP to R:
   "try k′ for h'_R"

   Upon receipt of such a message R uses k′ to decrypt em′, obtaining m.

4. TTP sends a message to S. The message depends upon the authentication option. In the case authoption″ is BothAuth or TTPAuth then TTP sends S:

   Message 4 (TTP auth), TTP to S:
   S(TTPSigKey,
   "I have released the key for S2TTP″ to R′")

   Otherwise, TTP sends S:

   Message 4 (no TTP auth), TTP to S:
   S(TTPSigKey,
   "I have released the key for S2TTP″")

   Note that in this last case, the receipt does not give evidence that S has not forged Message 2. (If the authentication option is SAuth then TTP only knows that it was provided an answer to a query which was generated by S; the source of the answer could be either R or S.) It may make sense to make this forgery slightly harder, for example, by adding the source IP address of Message 2 in the receipt. Note also that the receipt does not have to include k (or a hash of k) since S2TTP″ contains (an encryption of) this value.

5. Finally, if S receives a message of the same form as Message 4 then using TTPVerKey it verifies the signature, and checks that S2TTP″ matches one of the messages it sent. If these checks succeed then S notes that message m was received by R.

   Later on, if S ever wants to prove that R has received m to a judge, then S exhibits this message, em, k, q, and r, and the judge should check that these values and the TTP's public key match up.

3.3 Analysis

In this section we sketch an analysis of our protocol. We aim to state some of the main properties of the protocol, informally but clearly, and to provide informal arguments that these properties hold.

Throughout, we assume that messages may be intercepted and modified by a malicious party, following the standard conventions of the literature on security protocols, but assume that messages sent by TTP reach their destination. We focus on the modes of the protocol where TTP authenticates R. When only S authenticates R, inevitably, S is able to impersonate R and cause TTP to produce a certificate when R has not received the corresponding message.

There is plenty of interesting further research in this direction: writing a complete and formal specification of the protocol's properties, and constructing formal correctness arguments. While this work does not seem beyond the state of the art in protocol analysis (which has progressed significantly in the last few years), it remains quite considerable and difficult. It is beyond the scope of the present paper.

Proposition 1. R *cannot read* m *without* S *obtaining a receipt proving that* R *did so.*

Proof sketch: R cannot read m without first receiving a message containing em=E(k,m). It therefore cannot read m without receiving k from TTP. The non-malleability of the encryption function A ensures that the only way for R to obtain the key is to forward the field S2TTP' = S2TTP to TTP.[2] The protocol ensures that when TTP sends k to R it also sends a signed receipt to S. (We assume that both of these messages arrive at their destinations.) The receipt contains an authentication of R's identity (in a form depending on authoption), and a value S2TTP″ which is equal to the value S2TTP generated by S. This value is S2TTP = A(TTPEncKey, S | authoption | "give k to R for h_S"), where

[2] The non-malleability property is important since R serves as a communication channel between S and TTP. If A() was malleable it might have been possible for R to change the encrypted information, say from "give k to R for h" to "give k to R for h' ". This option enables R to send h' instead of h to TTP, and results in S not being able to present a valid receipt for the message m.

$\mathsf{h_S} = \mathsf{H}(\mathsf{cleartext} \mid \mathsf{q} \mid \mathsf{r} \mid \mathsf{em})$. S can therefore reveal m as well as k, q, r, and cleartext, and show that they correspond to the value signed by TTP.

PROPOSITION 2. S *cannot obtain a receipt proving that* R *read a message* m *without* R *being able to read this message.*

Proof sketch: Suppose that S has a receipt signed by TTP and claims that it proves that R read a message m. The receipt sent to S contains a value $\mathsf{S2TTP}'' = \mathsf{S2TTP} = \mathsf{A}(\mathsf{TTPEncKey}, \mathsf{S} \mid \mathsf{authoption} \mid$ "give k to R for $\mathsf{h_S}$"$)$, where $\mathsf{h_S} = \mathsf{H}(\mathsf{cleartext} \mid \mathsf{q} \mid \mathsf{r} \mid \mathsf{em})$. We assume for simplicity that there is only a single public key associated with TTP. (Otherwise the identity of the relevant key should be part of the signed receipt sent by TTP.) The decryption function of A is therefore deterministic and there is only a single possible decryption of S2TTP. Consequently, given the receipt, S can present only the values k, R, and $\mathsf{h_S}$ that were used by TTP.

Before generating the receipt, TTP received a value $\mathsf{h_R}$ from R and verified that it is equal to $\mathsf{h_S}$. The collision resistance of H ensures that R received the same encrypted value em that was used to generate $\mathsf{h_S}$. Since TTP sends the key k to R at the time it sends the receipt to S, and since we assume that the channel between TTP and R is reliable, R has both em and k. The message m that R can decrypt is therefore the decryption of em with k, which is exactly the message defined by the receipt.

3.3.1 Confidentiality

Although this is not its main focus, the protocol does provide confidentiality for the message m, both from external adversaries and (to some extent) from TTP. To observe this, note that one of our assumptions is that the communication channel between R and TTP is secure (e.g., using HTTP messages and SSL encryption). An external adversary can therefore only recover Messages 1 and 4 which contain encrypted information. If TTP follows the protocol then it too does not gain any information about the message, since it only learns the key k and the hash of the encrypted message.[3] If we assume, however, that the trusted third party TTP is corrupt, then it can eavesdrop on Message 1 which is sent from S to R. In this case it has the same information as R and can decrypt em and obtain the message m. The only way to prevent this attack is by adding an additional encryption layer to the message m, that can only be decrypted by R. If we do not assume the existence of a PKI then the only key that can be used for this encryption is the query/response pair (q,r) that S uses to authenticate R. Unfortunately, in the case where human users can memorize the response r, it would almost always be the case that the response does not contain sufficient entropy to provide security against off-line brute-force attacks by the corrupt TTP.

3.3.2 Further Discussion

Note that the authentication of R is important for two reasons. It is important for providing a receipt that identifies R as the receiver of the message, and for preventing someone who has hijacked Message 1 from contacting TTP and asking for the decryption key. Therefore, it is preferable that the authentication option is TTPAuth or BothAuth and there is a method of authenticating R to TTP. In this case TTP can provide a receipt that contains R's identity. Otherwise, if SAuth is the chosen authentication method and R is only authenticated by the query/response pair, then S can be convinced that R read the message but other parties cannot verify that a corrupt S did not impersonate R (communicating with TTP) in order to obtain the receipt.

There are several further noteworthy properties related to the computation of the hash values. First, it is important to put cleartext in the hash in order to prevent a corrupt S from sending a benign cleartext message to R, enclosing an incriminating message m. (Consider for example S that sends the cleartext message "please read the attached bill" but encloses an unrelated message which R would definitely not want to receive.) An unsuspecting R might read the cleartext message and decide to engage in the protocol, at the end of which S has a proof that R read the message. Making the cleartext message part of the receipt enables R to prove its innocence. Second, the query should be part of the hash in order to have better authentication of R. The response r alone is not sufficient since it might be the correct response for two different queries that identify different persons (e.g., "Wharton" might be the answer to the query "What is your mother's maiden name?" but also to the query "What business school did you attend?").

4. IMPLEMENTATION

The receiver software was the focus of the most important design decision for our implementation. We considered several approaches.

In one approach, the receiver downloads an email plug-in. Unfortunately, this approach limits the community of users who would be willing to use the system for two reasons. First, some users are reluctant or unauthorized to install software on their machines, especially in a corporate environment, thus would not use the system. Second, there are currently many different email readers that have significant market share. To be ubiquitous, an implementation would need plug-ins for all these email readers; this requirement makes the approach impractical.

Another approach uses a generic mail proxy that sits on the receiver's machine. The proxy could process all incoming and outgoing emails looking for those messages tagged as certified, and do the requisite processing of our protocol transparently. This is an attractive proposal since the proxy could perform many other useful tasks, such as filtering spam, and makes the system less dependent on the particular email reader being used. Indeed, we have implemented this approach for one version of the software that the sender uses. However, this approach still suffers from the requirement that the receiver must install a piece of software.

We took a third approach, which we now describe. Because of the shortcomings listed above, we decided to implement a receiver that uses only a standard MIME enabled email reader and a Java-enabled browser. This design decision dictated to a large extent how the rest of the system functions. However, we give the receiver the option of downloading a plug-in if they wish to have tighter integration with their mail system. Another design decision was to implement the protocol using Java. This was attractive because it fit in with our requirements on the receiver and because there are high-quality Java packages for handling

[3] We assume that this hash, $\mathsf{H}(\mathsf{cleartext} \mid \mathsf{q} \mid \mathsf{r} \mid \mathsf{em})$, does not reveal any information about the message em.

MIME email and cryptography operations, which simplified our coding.

Our implementation has two different sender programs; S can use either one of them. One provides a simple message composition window into which the user enters the sender's email address, the receiver's email address, the cleartext message, and the message to be certified. It also allows the setting of authentication options and other variations in the protocol. The second sender program is designed as a proxy: The user uses an ordinary email composition program, but instead of sending the message to the intended receiver, sends it to a special address that is redirected to the proxy. The proxy reads the real receiver from a special header field.

Both sender programs then proceed as follows. They form the first message in the protocol and encode it as a binary stream in an appropriate way. This binary stream is Base 64 encoded [10] into a textual string, call this string m1. Next, the sender programs form a two-part MIME email to the receiver. The first part is a plain text part explaining that the message is a certified email message and that it can be read by sending the second part to a Java-enabled browser. The second part is HTML containing an applet whose code (JAR files) will come from TTP's site and whose parameters describe the sender, describe the receiver, give TTP's URL, and give m1. The sender programs then send this email to the receiver.

When the receiver gets the email, they are shown the cleartext message and if they wish to read the certified email message they should double-click the HTML attachment to launch a browser. The browser downloads the applet from TTP's site and runs it with the parameters in the HTML. The applet Base 64 decodes m1, and, if necessary, obtains authentication information from the receiver. Then it forms the second message in the protocol and sends it by POSTing it (suitably encoded) to a cgi-bin script on TTP's site. To secure this POST message and its response, we build a secure channel using a combination of public-key and symmetric encryption. The cgi-bin script on TTP's site runs a Java application that reads the second message, and performs the necessary authentication and hash comparison checks. It returns a suitable piece of HTML that indicates either success or failure. In the latter case, the HTML also includes the third message of the protocol Base 64 encoded. The receiver applet then obtains the key from the returned third message and decrypts the original message from the decoded m1. It displays this message in its area of the browser's window. Currently, our implementation just displays the raw message in a scrollable text area. This could easily be extended to do some basic MIME processing. Finally, in case of success, TTP's application also sends the fourth message of the protocol to the sender as an ordinary email message.

The implementation was fairly simple—a straightforward combination of some standard packages and some code to glue it together. However, we ran into problems with the cryptography libraries. Java's standard libraries and JCE (Java Cryptographic Extension) provide a comprehensive range of cryptographic primitives. Many browsers, however, have older or even no versions of the libraries. The applet could include the libraries itself, except that most browsers do not allow applets to include cryptography code within the framework of the standard libraries (presumably in order to prevent Trojan Horse cryptography libraries from affecting the browser.) Thus, we were forced to re-implement these libraries as a stand-alone package not tied to the security mechanism. Similarly, we wanted to use SSL to secure the channel between the receiver and TTP, and while most browsers have SSL capabilities, these capabilities are not exposed to Java applets. Thus, we were forced either to implement SSL in Java ourselves or to resort to our own implementation of secure channels. We would recommend that designers of runtime systems think carefully about cryptography libraries, if possible making them standard and available across all platforms, and not preventing applets from extending the libraries with new methods for their own use.

The biggest advantage of our implementation is that it does not require the receiver to have anything more than a standard email reader and a Java-enabled browser—the special software needed to operate the protocol is downloaded through the standard mechanisms of the email reader and browser. A disadvantage of this scheme is that the original message is only available to the receiver in the applet—it is not amenable to the usual email-program functions. For example, it is more difficult to print, reply to, and search the email; and each time the user wishes to view the email, they must go through TTP. So while our solution is useful for initial deployment and avoiding the chicken and egg problem, ultimately an implementation must provide a more convenient and seamless solution. One compelling idea is to combine our approach with one of the other two approaches—proxy or plug-in. Our applet would enable receivers to receive and read certified emails without installing any software, but later they could install a proxy or plug-in for more convenience and seamlessness. Our applet could even facilitate the downloading and installation of the proxy or plug-in, and also of appropriate sender software.

5. ACKNOWLEDGMENTS

We thank Andrew Wright for his contribution to the early stages of this project.

6. REFERENCES

[1] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Symposium on Research in Security and Privacy*, pages 86–99. IEEE Computer Society Press, 1998.

[2] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. In Kaisa Nyberg, editor, *EuroCrypt 98*, pages 591–606. Springer-Verlag, 1998.

[3] G. Ateniese, B. de Medeiros, and M. Goodrich. Tricert: Distributed certified e-mail schemes. In *ISOC 2001 Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2000.

[4] G. Ateniese and C. Nita-Rotaru. Stateless-recipient certified e-mail system based on verifiable encryption. In B. Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 182–199. Springer-Verlag, 2002.

[5] http://www.authentica.com.

[6] Alireza Bahreman and J. D. Tygar. Certified electronic mail. In *Proceedings of the Internet Society Symposium on Network and Distributed System*

Security, pages 3–19, San Diego, CA, USA, February 1994.

[7] M. Blum. *Three applications of the oblivious transfer: Part I: Coin flipping by telephone; Part II: How to exchange secrets; Part III: How to send certified electronic mail*. University of California, Berkeley, CA, 1981.

[8] D. Boneh. Simplified OAEP for the RSA and Rabin functions. In J. Kilian, editor, *CRYPTO'01*. Springer-Verlag, 2001.

[9] D. Boneh and M. Naor. Timed commitments (extended abstract). In M. Bellare, editor, *CRYPTO'00*. Springer-Verlag, 2000.

[10] N. Borenstein and N. Freed. RFC 1521: MIME part one, September 1993. Section 5.2.

[11] http://www.certifiedmail.com.

[12] Richard Cleve. Controlled gradual disclosure schemes for random bits and their applications. In G. Brassard, editor, *CRYPTO'89*, pages 573–588. Springer-Verlag, 1990.

[13] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO'88*, pages 13–25, 1998.

[14] Ivan Damgård. Practical and provably secure release of a secret. In T. Helleseth, editor, *EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 200–217. Springer-Verlag, 1994.

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

[16] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *23rd Symposium on Theory of Computing*, pages 542–552. ACM, 1991.

[17] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[18] Matthew Franklin and Michael Reiter. Fair exchange with a semi-trusted third party. In Tsutomu Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 1–6, Zurich, Switzerland, April 1997. ACM Press.

[19] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. Simplified OAEP for the RSA and Rabin functions. In J. Kilian, editor, *CRYPTO'01*, pages 260–274. Springer-Verlag, 2001.

[20] O. Goldreich. Sending certified mail using oblivious transfer and a threshold scheme. Technical report, Computer Science Department, Technion—Israel Institute of Technology, 1984.

[21] B. Pfitzmann, M. Schunter, and M. Waidner. Provably secure certified mail. Technical Report RZ 3207 (93253), IBM, Research Division, Zurich, Switzerland, February 2000.

[22] http://www.postx.com.

[23] http://www.ietf.org/rfc/rfc2298.txt.

[24] V. Shoup. OAEP reconsidered. In J. Kilian, editor, *CRYPTO'01*, pages 239–259. Springer-Verlag, 2001.

[25] http://www.sigaba.com.

[26] http://www.zerogravitytech.com.

[27] J. Zhou and D. Gollmann. Certified electronic mail. In *Computer Secuirty—ESORICS'96 Proceedings*, pages 160–171. Springer-Verlag, 1996.

[28] http://www.zixit.com.