In Proceedings of the 1994 Network and Distributed Systems Security Conference, February 1994, pp. 3-19

CERTIFIED ELECTRONIC MAIL

Alireza Bahreman
Bell Communications Research
Piscataway, New Jersey

J. D. Tygar
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

We propose two new families of protocols for certified electronic mail. Certified electronic mail enables two mutually suspicious users to exchange a receipt for electronic mail. One family of protocols, the believers' protocols, use a trusted third party. The second family, the skeptics' protocols, use no third party. Our protocols are secure in a very strong sense; the probability of one party cheating can be made arbitrarily small. The protocols provide a practical example of the use of various innovative cryptographic techniques, including digital signatures, bit-commitment, and zero-knowledge interactive proofs. These protocols can be implemented in modern communication networks.

INTRODUCTION

Sue Sender wants to send electronic mail (e-mail) to Rob Recipient. Sue wants to obtain a receipt from Rob in return for her e-mail. Sue can send Rob Certified Electronic Mail (CEM). CEM supports ordinary e-mail with the addition that the recipient must sign a receipt in exchange for the received e-mail. If an independent third party is witness to the exchange, a proof of mailing will also be available for Sue. Both the receipt and the proof of mailing are dependent on the content of the e-mail received or sent. CEM is the digital equivalent of the post office providing certified mail (proof of mailing) and return receipt service for ordinary paper-based mail. However, both the receipt and the proof of mailing in the paper-based model are independent of the content of the letter. Dependency of the receipt and the proof of mailing on the content of the e-mail is one of the advantages of CEM over ordinary paper-based certified mail with return receipt.

CEM is an *end-to-end* service which can be implemented at the application layer of the International Standards Organization's (ISO) Open Systems Interconnect (OSI) Reference Model [17]. CEM can also be implemented on top of Privacy Enhanced Mail (PEM) [20, 19, 2, and 18]. In doing so, CEM users could enjoy *confidentiality*, *authenticity*, *integrity*, and *non-repudiation of message origin* in addition to *proof of mailing* and *non-repudiation of message receipt*. Briefly, confidentiality protects transmitted messages from unauthorized disclosure. Integrity ensures that the message content is not modified during transmission. Authentication is simply the assurance that the remote entity sending the message is correctly identified. Non-repudiation comes in two flavors, non-repudiation of message origin and non-repudiation of message receipt. The former protects against the sender denying transmission of the message while the latter protects against the recipient denying receipt of the message. Proof of mailing allows the sender to prove to any third party that the sender did in fact send the message to the recipient.

As part of a larger effort to introduce privacy enhanced mail within Bell Communications Research, the first author is planning to implement the Believers' CEM protocol, described in this paper. This prototype implementation will demonstrate the feasibility of the protocol in practice.

We present the requirements, common assumptions and our approach in implementing CEM protocols in the next section. This is immediately followed by the detailed description of our two approaches in providing CEM protocols. In our first approach, we use a trusted third party during the exchange of the CEM and the receipt. In the second approach, we carry out the exchange of the receipt for the CEM independent of any third party. We also suggest some applications of CEM followed by concluding remarks at the end of this paper.

TERMINOLOGY

We use the following terminology in this paper.

Conventional cryptosystems such as the Data Encryption Standard (DES) are also referred to as ***private-key*** or ***symmetric*** cryptosystems. There is a single key, simply called

the ***key***, used in the encryption/decryption process which must be kept secret between the communicating parties.

We use the terms ***asymmetric*** and ***public-key*** cryptography interchangeably to refer to cryptographic algorithms which use a pair of keys in the encryption/decryption process as appose to the single key used in conventional cryptosystems. The two components of the key pair are referred to as the ***secret*** and ***public*** keys.

SOLVING THE CEM PROBLEM

Requirements

Sue and Rob are *mutually suspicious* of each other. Either party may attempt to cheat at any time during the exchange of the receipt in return for the CEM. Sue would like to be able to receive a proof that she had indeed sent the CEM to Rob. Rob may want to receive the CEM without signing a receipt for it. Sue cannot force Rob to sign a receipt unwillingly. However, once willing, Sue wants to prevent Rob from being able to use her CEM until he signs for it. Rob on the other hand, wishes to sign a receipt only when he receives Sue's CEM. The receipt must be dependent on the content of Sue's CEM. This is to prevent either party from later claiming to have received or sent a different CEM. Only Rob should be able to generate receipts that are signed by him. Sue should be able to show the receipt to any third party and convince them that it is a signed receipt from Rob for the original CEM. Receiving receipts for past communication with Rob should not reveal anything to Sue with which she would be able to generate receipts for future communications. In our protocols, to violate the above conditions and cheat, one would need to either solve a computationally intractable problem or correctly guess a sequence of bits out of a large number of possibilities.

Common assumptions for CEM protocols

There are a number of underlying common assumptions in our protocols which are presented in this section. Each of the two protocols presented in this paper have additional assumptions which are listed in the appropriate section under that protocol.

I - Equal computational power and knowledge of algorithms

We assume that both the sender and the recipient have equal computational power and knowledge of algorithms.

II - Secure and privacy enhanced communication

All messages exchanged in our protocols need to be protected against eavesdropping by passive intruders and against tampering by active intruders. In other words, we need to secure the confidentiality and integrity of each message. This task can easily be accomplished by sending each message as a protected message using PEM. We therefore rely on the secure communication provided by the chosen underlying mechanism such as PEM. This assumption simplifies the presentation of the protocol.

III - All messages delivered in bounded time with no denial of service

A practical assumption is to place an upper bound on the expected message delivery time. This implies that all messages will eventually get delivered in bounded time. Should a message be lost or stolen off of the network, the originator will continue to resubmit the message until it is delivered. This way we avoid the denial of service problem, except when the network permanently fails. It is the responsibility of the originator to make sure that sent messages are delivered. The originator could, for example, request and wait for an acknowledgment. If the acknowledgment is not received after a specified time-out period, the message is retransmitted. The retransmission can also be done transparent to the sender at the transport layer of the communications network, or even at the application layer, using special user agent applications.

IV - Functional key distribution center

Users' signature should be universally verifiable. This requires a functional key distribution center for practical reasons. Any user on the network should be able to obtain the public component of another users' cryptographic key. The X.500 directory services [7], recommended by the Consultative Committee on International Telephony and Telegraphy (CCITT), promises to provide the means for the required key distribution center. The required public key distribution and retrieval can also be achieved through extensions of the Domain Name System (DNS), or creation and use of socket-based and/or mail-based certificate responders.

V - One-way functions exist

A fundamental assumption made is that one-way functions do exist. One-way functions are functions that can be easily computed but are difficult to invert. This assumption has not yet been proven. However, functions do exist that are hard to invert and are conjectured to be one-way. These include factoring a large composite number (a product of two large primes), and discrete log modulo a large randomly chosen prime [15]. Several assumptions follow directly from Assumption V which we will state as corollaries below:

A - Secure pseudo-random generators exist

B - Secure private-key or symmetric cryptosystems exist

C - Secure digital signature schemes exist

The existence of one-way functions is a necessary and sufficient assumption for the existence of secure pseudo-random generators [15], secure private-key cryptosystems [11], and secure digital signatures [25]. These secure cryptographic primitives are the building blocks of our protocols. For a discussion on the limits of the provable consequences of one-way functions see [27 and 16].

Approach

In general, the CEM problem makes sense only in the presence of at least three parties—the sender, the recipient, and a third party. If only Sue and Rob existed in the world, it makes no sense for Sue to require Rob to sign a receipt for her CEM. The receipt is useless unless there is a third party which can verify the receipt and hold Rob responsible for his signature. The third party settles any disputes that may arise during or after the exchange. There can be more than one third party playing different roles during and after the exchange. While the existence of a third party is necessary by definition, its nature of presence and amount of involvement may vary. For example we distinguish between on-line computerized third parties (or server daemons) which are available over the communication network and off-line human judges that can act as a third party. The amount of involvement of a third party also varies from fully involved to not involved during the exchange. In the former case the third party is essential during the exchange, while in the latter case the third party will only enforce the legal consequences after the exchange has taken place.

In the absence of a trusted third party, the CEM problem is related to *exchange of secrets* and *contract signing*, both of which are examples of *symmetric exchange* problems. In symmetric exchange problems, there are two parties to the exchange. Both parties in the exchange have information of *equal value* to exchange. The term "equal value" is used loosely and can apply to a wide range of applications. In contract signing, for example, the information of equal value are the signatures of each party to the contract. Both parties need to verify the information they receive from the other party. *Gradual* and *verifiable* release of information can *approximate* a solution to symmetric exchange problems. By gradual we mean that both parties divide their respective information into small parts. They then take turns in a serial exchange of these parts. Without loss of generality, consider the example in which each part is one bit. Each bit should be verifiable by the other party to avoid cheating by sending junk bits. We say that this approximates a solution since the first party to begin the serial exchange is always in a disadvantage in having revealed one bit of information more than the other party. This may result in a 2:1 computational disadvantage for the party who started the exchange. If there exists a polynomial-time algorithm to deduce the missing bits of the information from the partial information provided in the middle of the exchange, both parties can obtain the other party's information. The disadvantaged party must guess the value of one extra bit and perform the same polynomial-time algorithm twice leading to the 2:1 computational factor. Tedrick [28 and 29] has demonstrated how two adversaries can exchange a "*fraction of a bit*". Using Tedrick's method, the advantage of one party over the other could be made arbitrarily small. He has shown how to achieve a $(2^n + 1):2^n$ expected advantage factor for any integer n. For example, by choosing $n = 2$ we can achieve a 5:4 expected computational advantage instead of the ordinary 2:1 advantage we get by exchanging one bit at a time.

Providing a solution to the symmetric exchange problem is difficult. This is because a *verifiable simultaneous exchange* is required. Not only the information parts from both parties needs to be exchanged simultaneously, each part must be verified at the same time. Consider a case in which Sue and Rob each have a secret key of equal length in bits to exchange. Even if they can simultaneously exchange their keys, one bit at a time, either party can cheat by sending junk bits. Therefore, unless the verification occurs during the simultaneous exchange, one party can gain an advantage over the other by cheating. Simultaneity is very impractical, if not impossible, to achieve in practice and simultaneous verification would seem to be even harder to achieve. We therefore have no choice but to rely on the approximate solution to symmetric exchange—gradual and verifiable release of information.

The CEM problem is an *asymmetric exchange* problem. There are two reasons for this asymmetry, both of which are due to the dependency of the receipt to the content of the message. First, Sue must send Rob her message before she can ask for a receipt. Second, only the sender needs to verify the information received from the recipient. In the absence of a trusted third party, the CEM asymmetric exchange problem can be transformed into a symmetric exchange problem for which an approximate solution exists using gradual and verifiable release of information.

In the presence of a trusted third party, called the *postmaster*, CEM asymmetric exchange can be transformed into an *atomic operation*. Sue submits her CEM to the postmaster. The postmaster asks Rob for a receipt for Sue's CEM. Once the postmaster has received and verified the receipt, it needs to send the receipt to Sue and the original CEM to

Rob. Ideally, this needs to be done in one atomic operation where both parties receive their respective information simultaneously. Otherwise, one party may have an advantage over the other by receiving the other's information early. The period in which one party has an advantage over the other is considered to be a *vulnerability* period.

An approximate solution to the atomic operation problem can be reached by minimizing the vulnerability period. The importance of the length of the vulnerability period is highly dependent on the application in which CEM is used. Providing a solution to the atomic operation problem is difficult.

We distinguish between and classify two families of protocols implementing CEM based on the nature of presence and amount of involvement of third parties during the exchange of CEM for a receipt. The first family of protocols use an on-line trusted third party during the exchange. Because we must believe in the trustworthiness of a third party, we refer to this class of protocols as the Believers' CEM, abbreviated as B-CEM. The second family use no third party during the exchange. Since they do not assume the existence of a trusted third party during the exchange, these protocols are referred to as Skeptics' CEM abbreviated as S-CEM. We discuss these two types of CEM protocols next.

BELIEVERS' CEM (B-CEM)

Overview

The B-CEM protocols are characterized by the active use of a trusted third party, called the *postmaster*. The postmaster acts as an independent agent arbitrating the exchange of a receipt from the recipient (Rob), for the CEM from the sender (Sue). Conceptually, there are five major phases in any B-CEM protocol. The interactions between the parties in the protocol is depicted in Figure 1. In phase one, Sue prepares the CEM and sends it to the postmaster. In phase two, the postmaster will send the proof of mailing to Sue if required. Also in phase two, the postmaster uses symmetric or private-key encryption to encipher the CEM using a randomly generated cryptographic key producing a ciphertext. The postmaster then stores a record of the key, the CEM, sender-recipient information, and any other information to uniquely identify the transaction. The postmaster is then responsible for transmitting the ciphertext to Rob. In phase three, Rob signs a receipt corresponding to the received ciphertext and sends the receipt to the postmaster. In phase four, the postmaster verifies the receipt and if valid stores a copy of the receipt. At this point the postmaster has all the needed information to begin the exchange. The postmaster begins a two part operation. First the postmaster sends the cryptographic key to Rob to decipher the ciphertext. Next, the postmaster sends a copy of the receipt and the cryptographic key to Sue as the return receipt. In phase five, both parties individually verify the information they received. Note that all messages exchanged need to be protected for confidentiality, authenticity, integrity and non-repudiation of message origin (Assumption II). The sender-recipient information must be duplicated inside the message body to protect it against tampering.

Notation

We cover the notation used to describe the protocol here. We present the protocol in a three column format with a special font as shown in Figure 2. The first column indicates where an action is taking place, or in the case where a message is being exchanged, between which parties. The second column outlines the action, or the message being exchanged. The third column, or the line number, is used to uniquely identify each line of the protocol. The number is a combination of the figure number and a line number within that figure. All messages and strings are represented in lower case letters while the sender, the recipient, and the postmaster are represented in upper case letters and abbreviated as `S`, `R`, and `PM` respectively. For example, Figure 2 represents the case when Sue sends a CEM to the postmaster.

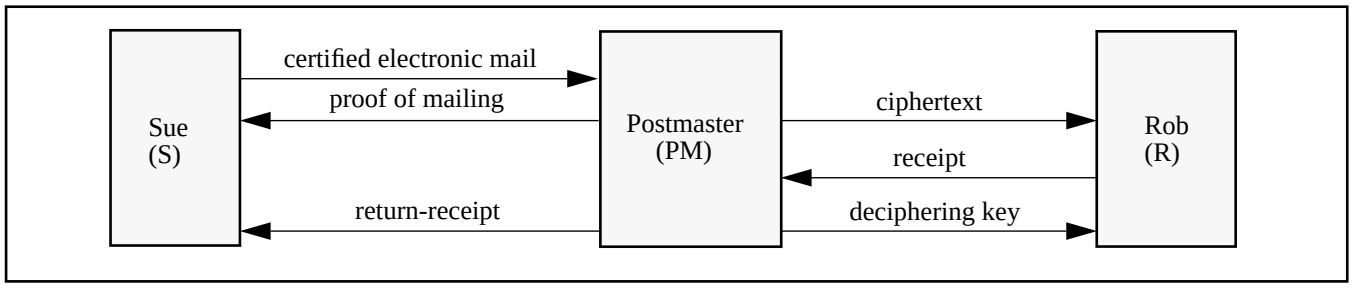

Figure 1. High-level overview of the interactions among the parties in a B-CEM protocol.

```
S-->PM cem                                    2.1
```

Figure 2. Sample notation. Sue sending cem to the postmaster.

The symbol ‘:=’ represents the assignment operator; the right hand side expression is evaluated first and the result is assigned to the left hand side. The angle braces, ‘<’ and ‘>’, are used to indicate the concatenation of strings.

Functions or processes used to manipulate messages and strings are capitalized. Figure 3 illustrates symmetric encryption using a randomly generated key. For example the `Random-Key-Generator` is a secure pseudo-random number generator used to generate a random `key` by the postmaster (3.1). The key, `key`, is then used along with the `Encipher` function to encipher Sue’s CEM, yielding `ciphertext` (3.2). The `ciphertext` can then be deciphered by Rob using the same `key` and the `Decipher` function (3.3). Both the `Encipher` and `Decipher` operations are examples of a secure symmetric or private-key cryptosystem.

```
PM key:=Random-Key-Generator()        3.1
PM ciphertext:=Encipher(cem,key)      3.2
R  cem:=Decipher(ciphertext,key)      3.3
```

Figure 3. Sample notation. Private-key encryption.

We use the `Sign` and $\texttt{Sign}^{-1}$ functions to represent a secure digital signature scheme (asymmetric cryptosystem). Figure 4 illustrates the notation. These functions are applied to messages along with the users’ secret or public key components. The keys are respectively represented as `secret-key` and `public-key` to distinguish them with the key used in symmetric cryptosystems (`key`). We do not specify how each user in the protocol obtains others’ public key. This can be done by using a key distribution center (Assumption IV). To indicate possession of a key, the users’ acronym is prefixed to the name of the key. For example, Sue can sign her e-mail using her secret key (4.1). In reality, a message digest or hash value of the e-mail is signed instead of the e-mail itself. All signatures can be verified by applying the $\texttt{Sign}^{-1}$ function to the signature (4.2). In reality, the verification may involve computing the message digest and comparing it with the output of the $\texttt{Sign}^{-1}$ function.

Other functions used are the `Store` and `Compare` functions. The `Store` function is used to maintain information in private and protected databases. This is particularly needed by the postmaster to maintain a record of CEM

```
S signature:=Sign(e-mail,
                  S-secret-key)       4.1
R e-mail:=Sign^-1(signature,
                  S-public-key)       4.2
```

Figure 4. Sample notation. Signature processing.

transactions. Access to databases of such records should be strictly controlled. The actual details of database operations are not specified in our protocol. Only the postmaster’s storage activity is outlined as it is critical to the security of the protocol. The `Compare` function is used to compare the equality of information and is used in verifying the validity of received messages. If the `Compare` function or any other function ever fails because of a security problem (for example, the verification of an invalid signature) or any other reason, the process executing the function will abort the protocol. In what follows, we outline each phase of the protocol in more detail.

Protocol

In this section, the five phases of a B-CEM protocol are presented.

Every message exchanged among the parties in the protocol consists of a content section (or body) and a header section. Obviously, appropriate header information is needed for every message, but we do not present the full details here. (For example, we suggest the use of the printable string “multipart/B-CEM” as the value of the *Content-Type* field of the header in CEM. This would allow Multipurpose Internet Mail Extension (MIME) capable mail user agents to be used to interface with users.) Also included in the body is an inner header which holds the sender-recipient information for Sue and Rob. This is needed for two reasons. First, it would help the postmaster obtain needed information. Second, since the message body is protected against tampering, the integrity of the inner header can be assured.

It is important to recall that every message sent on the network is assumed to be protected against tampering and/or eavesdropping by an intruder (Assumption II). Hence, only the intended recipient can obtain the content of a received message and verify its integrity. The confidentiality of each message sent could be achieved by the sender asymmetrically encrypting the message with the public key of the intended recipient. Alternatively, one could use PEM which uses a hybrid of symmetric and asymmetric cryptographic techniques to offer confidentiality, authenticity, integrity and non-repudiation of message origin.

Phase one

In the first phase (see Figure 5) Sue signs her e-mail message and sends the result to the postmaster.

```
S cem:=Sign(e-mail,S-secret-key)5.1
S-->PM cem5.2
```

Figure 5. Phase one of a B-CEM protocol. Cem generation.

Phase two

The postmaster verifies Sue's signature. It is important for the postmaster to store a record of this transaction at this point in *stable storage* (memory that retains information despite failures). The record includes the sender-recipient information, a timestamp (for replay prevention), and the text of Sue's CEM. This record can serve as proof of mailing for Sue. However, if requested, the postmaster can sign Sue's CEM, `cem`, and return it to Sue as her proof of mailing. The postmaster then generates and stores a pseudo-random number, `PM-key`, with which it enciphers `cem`. The resulting ciphertext (`ciphertext`) is signed by the postmaster and sent to Rob. Rob cannot decipher `ciphertext` without `PM-key`. Rob must sign a receipt to obtain `PM-key`. It is preferable to have the sender receive the proof of mailing at the same time the recipient receives the enciphered message. The postmaster's actions in phase two are detailed in Figure 6.

Phase three

Once `cipher` is received, Rob verifies the postmaster's signature and obtains and stores `ciphertext`. Rob then signs `ciphertext` to produce the receipt, `receipt`. Note that since Rob does not yet know anything about the CEM content, he may not be able to use `cipher` alone to verify the postmaster's signature. Some other technique may be required here. For example, the postmaster can append its identity to `ciphertext` before signing it. If the postmaster's identity is recovered after applying Sign^{-1} to `cipher`, the signature is deemed valid. In reality, this problem will be eliminated if signatures are constructed using the message digest. Rob's processing is detailed in Figure 7.

```
R ciphertext
     :=Sign^-1(cipher,
               PM-public-key)        7.1
R receipt
     :=Sign(ciphertext,
            R-secret-key)            7.2
R-->PM    receipt                    7.3
```

Figure 7. Phase three of a B-CEM protocol. Receipt generation.

Phase four

Figure 8 presents the postmaster's actions in phase four. In order to check the validity of the `receipt`, the postmaster compares the original `ciphertext` (see 6.5) with that resulted by applying Sign^{-1} to `receipt` (8.1). If the verification fails, the postmaster aborts the protocol and stops. If all succeeds, the postmaster appends `receipt` to the stored record (8.2, see also 6.3 and 6.6). This is useful for future inquiries and can be used as Sue's return receipt. Should a dispute occur between the parties, the record maintained by the postmaster can be presented to a judge. The postmaster then prepares two messages. The first is for Rob and includes `PM-key`. The second is for Sue and contains both `PM-key` and `receipt`. The postmaster then signs and sends the two messages. It is

```
PM      e-mail          := Sign^-1(cem,S-public-key)                    6.1
PM      proof-of-mail   := Sign(cem,PM-secret-key)                      6.2
PM      Store(<sender-recipient information,cem>)                       6.3
PM      PM-key          := Random-Key-Generator()                       6.4
PM      ciphertext      := Encipher(cem,PM-key)                         6.5
PM      Store(PM-key)[a]                                                6.6
PM      cipher          := Sign(ciphertext,PM-secret-key)               6.7
PM-->R  cipher                                                          6.8
PM-->S  proof-of-mail                                                   6.9
```

a. The randomly generated key, `PM-key`, is appended to the record stored for this CEM transaction.

Figure 6. Phase two of a B-CEM protocol. Postmaster receives cem and forwards to recipient.

```
PM       Compare(ciphertext,Sign^-1(receipt,R-public-key))          8.1
PM       Store(receipt)                                             8.2
PM       deciphering-key:=Sign(PM-key,PM-secret-key)                8.3
PM       return-receipt:=Sign(<receipt,PM-key>,PM-secret-key)       8.4
PM-->R   deciphering-key                                            8.5
PM-->S   return-receipt                                             8.6
```

Figure 8. Phase four of a B-CEM protocol. Postmaster verifies receipt and begins the exchange.

preferable to have the two messages sent in a single atomic operation. Atomicity provides for a fair exchange without granting one party advantage over the other. The goal is to have Sue receive `receipt` at the same time Rob receives `PM-key`. Otherwise, one party would receive its respective information early. In such cases, the information stored by the postmaster could be used to resolve any dispute. Recall that the postmaster's database is considered to be stable storage. If either party does not receive the valid information, they may request the postmaster to send it to them again.

Phase five

After verifying the postmaster's signature, Rob uses the `PM-key` received to decipher `ciphertext`. If `PM-key` was not received after a specified time-out period or if the received signature was invalid, Rob will repeat requesting the key from the postmaster by resubmitting `receipt` (see 7.3). The validity of `cem` can easily be determined by verifying Sue's signature. However, Rob need not worry if Sue's signature is invalid. As the receipt is dependent on the content of the `cem` received, Rob only signed for what he received and cannot be held accountable for what he did not receive. See Figure 9 for the outline of Rob's activity during phase five.

```
R  PM-key
     :=Sign^-1(deciphering-key,
               PM-public-key)          9.1
R  cem
     :=Decipher(ciphertext,PM-key)     9.2
```

Figure 9. Phase five of a B-CEM protocol. Recipient verifies information.

Sue also verifies the postmaster's signature (see Figure 10). If invalid or if Sue never received a response, she can request the postmaster to retransmit. Sue can also check the validity of `receipt` by forming a ciphertext by enciphering `cem` with `PM-key` and comparing the ciphertext just obtained with that generated by applying the `Sign^-1` function to `receipt`. Sue is not required to verify the receipt as the postmaster has already done that in phase four (see 8.1).

```
S <receipt,PM-key>
      :=Sign^-1(return-receipt,
                PM-public-key)       10.1
```

Figure 10. Phase five of a B-CEM protocol. Sender receives receipt.

Both Rob and Sue can store relevant information for their records at this time for future use.

Assumptions unique to B-CEM protocols

VI - Trusted and secure third parties are available to all

In addition to the common assumptions enumerated before, B-CEM protocols, by definition, require a trusted and secure third party (the postmaster). The postmaster arbitrates a *fair* exchange of Sue's CEM in return for Rob's receipt. The postmaster will not modify or disclose messages or the cryptographic key used in the protocol. It will also make its services available to all legitimate users. The postmaster functions correctly (as specified in the protocol). If the postmaster is made available on-line without human interaction, its secret key will be needed on the system. In such a case, precautions must be made to secure access to that key against intruders. We assume that the combination of physical security and trusted human operators would provide adequate protection. The use of token devices such as smartcards could also assist in securing the postmaster's secret key.

VII - Stable storage with strict access control mechanism

The postmaster is also assumed to use a stable storage for maintaining records of CEM transactions. The use of stable storage helps solve problems arising from network or process failures. Strict access control to this database is assumed to be enforced.

Discussion

It must be noted that there can be different implementations of our B-CEM protocol. We have presented one such example which we felt most completely and clearly presents our ideas. Developers implementing the protocol face trade-offs and must address specific requirements. Issues of practicality, cost, performance, and complexity must be considered. Unfortunately there is no free lunch; improvements come at a cost of increased complexity. We informally discuss some other issues of relevance in this section.

Implementing the postmaster

Theoretically, only a single postmaster is required by the protocol. However, storage requirements, processing power, and bandwidth limitations may necessitate more than one postmaster to be available. Since there can be arbitrarily many postmasters, we do not believe that any one postmaster would be overwhelmed with requests. In any event, since the postmaster is used for every CEM exchange, it represents a bottleneck. This bottleneck can be alleviated by reducing the storage, bandwidth, and processing requirements of the postmaster. For example, the storage requirements can be reduced by storing the message digest instead of the message itself. Message digests can be obtained by applying a one-way hash function [23 and 24] to the message. One must use a secure hash function which is hard to invert. Otherwise, the sender (or the recipient) may be able to cheat by claiming to have sent (or received) a different message. Simple timestamping of the message before applying the hash function would also help make the digest uniquely identify the message (for example, if the same message is being sent more than once). The hash can then be used to facilitate retrieval of records pertaining to a particular CEM transaction. Methods to alleviate the postmaster's storage requirements come at the expense of additional administration in the protocol to retrieve records of CEM transaction.

Some long-term bookkeeping is required of the postmaster agents in B-CEM protocols. The postmaster is required to maintain a record of the exchange until a copy of the record signed by the postmaster is sent and received by both parties. This eliminates problems that may arise from one party failing during the vulnerability period or the communication network failing during the exchange. In the model presented here, the postmaster maintains a record of the transaction which can be used by the sender as proof of mailing.

Each postmaster should be built as a tamper-resistant module which can be trusted. The postmaster must function correctly as specified in the protocol. Problems may arise if the postmaster's storage facility is compromised, since the postmaster maintains records of old CEM exchanges.

Probability of undetected cheating

With a trusted third party, the probability of undetected cheating by either party becomes negligible; but this occurs only after the trusted postmaster has received and verified all the information that needs to be exchanged. (By *cheating* we mean Sue obtaining the receipt while Rob never receives the CEM, or Rob obtaining the CEM without signing a receipt for it.) We have used a secure private-key cryptosystem as a building block to our B-CEM protocol and have assumed a reliable communication channel using PEM. The security of our protocol relies on the security of these building blocks. The probability of undetected cheating is directly associated with the security parameters of the building blocks. For example, in order for Rob to cheat, he would need to reliably decipher `ciphertext` without `PM-key` (see 6.5). Assuming a secure symmetric cryptosystem (Corollary B), the probability of his doing so in tractable time is negligible. As another example, Sue might try to forge Rob's signature to cheat. This contradicts Corollary C.

Cryptographic ambiguity

In phase three, Rob blindly signs ciphertext. It is ambiguous at this point what that ciphertext represents. However, Rob need not worry, as his signature on the ciphertext is in the context of this CEM transaction. If challenged, Rob can supply `cipher` (see 6.8), which is signed by the postmaster, to justify having signed the enclosed ciphertext (see 6.7). He cannot be held accountable if the ciphertext happens to be something he would not otherwise be willing to sign. In addition, the ciphertext is created by a trusted postmaster with a randomly chosen cryptographic key. The probability that the ciphertext would actually be something that Rob would not want to sign is negligible as it highly depends on the random key chosen from a large number of possible keys. Furthermore, no one has an *a priori* knowledge of this random key.

Can Sue use Rob's receipt on a message m_1 for another message m_2? The probability of Sue being able to do this is negligible in tractable time. The `return-receipt` obtained by Sue (8.6) is signed by the postmaster and includes both `PM-key` and `receipt` (8.4). The postmaster's signature binds the randomly chosen `PM-key` used in generating `ciphertext` to Rob's `receipt`. In this way, Rob's `receipt` is dependant on the content of the original `cem` (see 6.5). For Sue to convince someone that Rob really signed for message m_2 instead of m_1, she

should, after the CEM transaction is over, create a message m_2 which when symmetrically encrypted using PM-key would result in the same ciphertext as ciphertext (6.5). Her probability of success is negligible assuming a secure symmetric cryptosystem (Corollary B).

SKEPTICS' CEM (S-CEM)

Background and notation

We use cryptographic techniques such as *bit-commitment* schemes and *zero-knowledge interactive proofs* in implementing the S-CEM protocol. We therefore briefly discuss both techniques and enumerate our requirements in this section. We also describe the notion of a *signed commitment* which we use in our S-CEM protocol.

Bit-Commitment Schemes

We extend the notation presented earlier to include the concept of a commitment to a cryptographic key. We use secure bit-commitment schemes to commit to individual bits of the key. The commitment to any key is then the ordered set of bit-commitments to all bits of the key. Several bit-commitment schemes have been introduced in the literature [4, 5 and 9]. Usually, a commitment to a bit *b* is computed as a function of *b* and a random number *r*. Revealing the commitment can then be achieved verifiably by releasing the value of the random number *r* used in computing the commitment. We use the phrase revealing (or releasing) commitments to imply opening commitments. Without referring to a specific commitment scheme, we use the following notation to represent a commitment scheme. A pair of functions Bit-Commitment, and $\text{Bit-Commitment}^{-1}$ is used to create commitments and to reveal previously committed bits respectively.

Consider the case where Sue commits to the value of a random key S-key and later reveals her commitment to Rob depicted in Figure 11. Sue's commitment is the ordered set of bit-commitments for every bit of S-key (11.1). We use curly braces, '$\{$' and '$\}$', to indicate the ordered set; the subscripted number after the right brace indicates the lower bound of the ordered set while the superscripted number represents the upper bound of the ordered set. Sue later reveals her commitment by releasing the random numbers she used in generating the commitments. To do this, Sue iterates for |S-key| times (number of bits of S-key). In each iteration, she releases the random number S-r-i, which she used to commit to S-key_i (11.2). Where S-key_i and S-r-i represent the *i*th bit of S-key and Sue's *i*th random number respectively.

$$\text{S commitment} := \{\text{Bit-Commitment}(\text{S-key}_i, \text{S-r-i})\}_{i=1}^{|\text{S-key}|} \quad 11.1$$

For (i = 1 to |S-key|) Do 11.2

S-->R S-r-i

Figure 11. Sample notation. Bit-commitment.

Rob can verify each *i*th random number received by applying $\text{Bit-Commitment}^{-1}$ to the commitment of S-key_i (Figure 12). The function $\text{Bit-Commitment}^{-1}$ fails if an invalid S-r-i is sent. In this way, cheating can be detected.

$$\text{R } \text{S-key}_i := \text{Bit-Commitment}^{-1}(\text{Bit-Commitment}(\text{S-key}_i, \text{S-r-i}), \text{S-r-i}) \quad 12.1$$

Figure 12. Sample notation. Obtaining the committed bits.

Properties of a secure bit-commitment scheme are also enumerated in the literature [5]. Informally, it must be very difficult for anyone, who does not know *r*, to find *b* from the commitment with a better than 50 percent chance. The person computing the commitment should not be able to reveal a committed bit as both 0 and 1. More formally, a secure bit-commitment scheme used by Sue would have the following properties:

- Sue can commit to any bit *b* in tractable time;
- By releasing her bit-commitment, Sue can convince another user that the bit she committed to is indeed *b*. Sue must not be able to reveal a bit-commitment as both *b* and (1-*b*);
- Rob or any other user cannot learn anything from the way in which Sue creates and reveals her bit-commitments even after several similar interactions with Sue;
- There is no tractable function on Sue's bit-commitment correctly giving the value of her committed bits better than half the time.

Signed commitment

We use the commitment scheme described above in a signed message which we call a signed commitment. A signed commitment combines a bit-commitment scheme with a signature scheme. Signed commitments are used in

our S-CEM protocol. We now formally define a signed commitment.

Definition 1 - A signed commitment

A signed commitment of a user, identified by U, on a message, m, is a message signed by U consisting of a ciphertext and a commitment to the cryptographic key used to generate the ciphertext (see step 13.4 in Figure 13). The message, m, is signed by U to form U's signature of m (13.1). The ciphertext is the output of a symmetric encryption algorithm applied to U's signature of m using a cryptographic key, k, of length |k| bits (13.2). The commitment is an ordered set of bit-commitments, one for each bit of k. The bit-commitment for each bit of k is created using an independent random number U-r-i (13.3).

Zero-Knowledge Interactive Proofs (ZKIPs)

The complexity class NP is well known. One could view NP as the class of languages *L*, which have polynomial-time *proof-systems* for language membership. The proof-system consists of two communicating parties, the *prover* and the *verifier*. The verifier is a polynomial-time machine which checks the correctness of the proof. The prover may either be computationally more powerful than the verifier or possess some additional knowledge or both. On every input string $x \in L$, the prover proves to the verifier that *x* is in fact in *L* by submitting a *certificate* to the verifier. The certificate must have a length polynomial in the length of *x*. The interaction is very simple and consists of the prover sending the certificate to the verifier. If $x \notin L$, the prover cannot obtain any certificate which could convince the verifier of the membership of *x* in *L*.

Goldwasser, Micali, and Rackoff [12 and 13] extended the classical notion of NP proof systems by incorporating randomness and by allowing a more complex interaction. The randomness is achieved by allowing the communicating parties to flip unbiased coins resulting in what they termed "probabilistic version of NP". The complex interaction is achieved by having the prover and the verifier exchange multiple messages instead of the one message ordinarily exchanged in NP proof-systems. Due to the randomness introduced, the verifier is allowed to err with a small probability. In other words, the verifier may erroneously accept the proof of language membership for some very small number of inputs which are not an element of the language. The probabilistic version of NP is called an *interactive proof-system*. Let IP be the class of languages that admit an interactive proof-system. It is clear that NP covers a subset of the languages in the class IP; every $L \in$ NP has an interactive proof-system in which the verifier never errs and the interaction involves the sender sending only one message (the certificate) to the verifier.

To better understand the notion of ZKIP, it is necessary to describe interactive proof-systems in greater detail. Throughout this paper, unless stated otherwise, by ZKIP we refer to zero-knowledge interactive proofs of language membership. We repeat the terminology and definition presented in [13]. An *Interactive Turing Machine* (ITM) is a Turing machine with five tapes; a read-only input tape, a work tape, a read-only random tape, one read-only communication tape, and one write-only communication tape. The random tape contains an infinite sequence of random bits, and can only be scanned in one direction. Flipping an unbiased coin can be simulated by the ITM simply reading the next bit on its random tape. An *interactive protocol* is an ordered pair of ITM's *P* and *V* such that they share the input tape. Furthermore, *P*'s write-only communication tape is *V*'s read-only communication tape and vice versa. This way, the two ITM's can communicate. The two machines take turn in being active, with *V* being active first. Either machine can terminate by not sending any message in the active stage. Machine *V* accepts (or rejects) the input by outputting *accept* (or *reject*) and terminating the protocol. The sum of all computations of each machine is its total *computation time*. Machine *V*'s total computation time is bounded by a polynomial in the length of the common input. While machine *P* is assumed not to be computationally bounded in [13], we relax this requirement for practical reasons. We simply require that machine *P* be more powerful than machine *V* (either computationally, or possess additional information). Figure 14 is reproduced from [13] and illustrates an interactive protocol.

An interactive proof of language membership for a language *L* is an interactive protocol in which two properties hold: *Completeness* and *Soundness*.

U	signature	:= Sign(m,U-secret-key)	13.1
U	ciphertext	:= Encipher(signature,k)	13.2
U	commitment	:= $\{\text{Bit-Commitment}(k_i,\text{U-r-i})\}_{i=1}^{\lvert k\rvert}$	13.3
U	signed-commitment	:= Sign(<ciphertext,commitment>,U-secret-key)	13.4

Figure 13. Sample notation. A signed commitment.

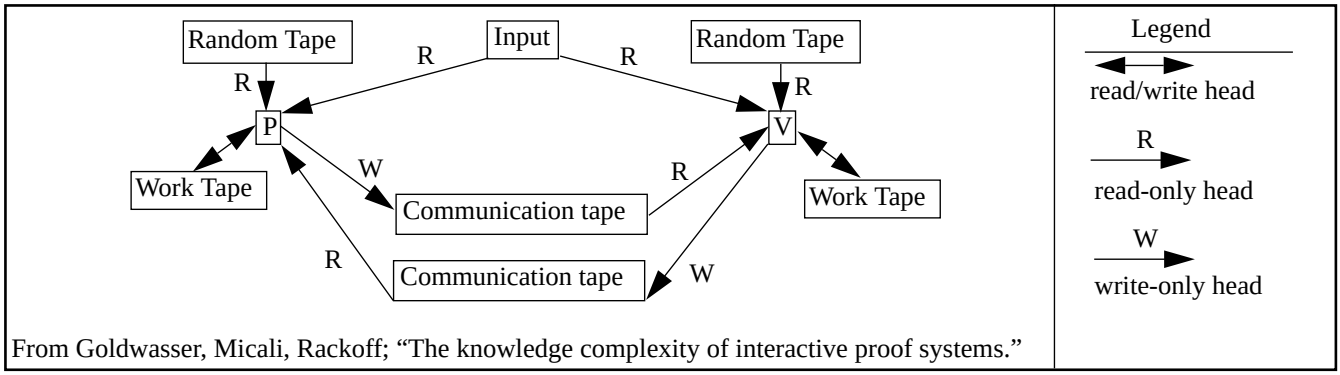

Figure 14. **An Interactive Protocol**

Completeness - If $x \in L$, at the end of the interactive protocol *V* accepts with overwhelming probability.

Soundness - If $x \notin L$, no machine acting as *P* can convince *V* to accept (except with negligible probability).

Notice that the probability of error, ε, can be made arbitrarily small. In [13], ε was set equal to $|x|^{-k}$ for any positive integer *k*. It was also mentioned that by using standard techniques, the error probability could be arbitrarily reduced to any small number (for example, $2^{-|x|}$). Of course the goal of a secure interactive proof system is to have ε as small as possible.

What a verifier sees in the course of the communications constitutes a *view.* This view includes the verifier's random tape, the input tape, and the two communication tapes. In other words, a view consists of the input string, the verifier's coin tosses, and the content of the conversation with the prover. Due to the probabilistic nature of the random tape, a probability distribution is defined on this view. For more detail and formal discussion, please refer to [13]. At this point, we are ready to discuss ZKIP.

An interactive proof of language membership for a language *L* is *perfect zero-knowledge* if the verifier learns only one bit of information (namely, whether or not the input belongs to *L*). More formally, an interactive proof of language membership is perfect zero-knowledge if for each polynomial-time verifier, and the corresponding view *v,* there exists a polynomial-time *simulator* capable of producing the same view *v,* without ever talking to the prover. Intuitively, since the view can be generated by a simulator, the verifier has not learned anything from the prover that the verifier could not have calculated alone.

An interactive proof of language membership is considered *statistically zero-knowledge* if the view created by the simulator, *v′*, is *statistically indistinguishable* [13] from *v.* An interactive proof of language membership is considered *computationally zero-knowledge* if the view created by the simulator, *v′*, is *computationally indistinguishable* [13] from *v.* Informally, computational indistinguishability implies that the simulator would require more than polynomial-time to simulate the interaction. Statistical indistinguishability, on the other hand, implies that the simulated view generated is statistically close to, but not equal to, the original view. A probabilistic polynomial-time verifier cannot learn any information in a computational ZKIP except for whether or not the input is in the language. A statistical ZKIP implies the same, but this time for an arbitrary powerful verifier.

Overview of the S-CEM protocols

One can implement a protocol offering CEM without using a trusted third party. We introduce a method for achieving this goal using an interactive protocol. Unlike B-CEM, S-CEM protocols do not rely on the presence of a trusted third party during the exchange of CEM in return for a receipt. Both parties should be available on-line in order to engage in an interactive dialogue. This differs from the store-and-forward mechanism used in ordinary e-mail. Since there is no third party to settle any dispute that may arise during the exchange, the protocol is designed in such a way that two properties hold: First, with high probability, neither party can cheat the other and escape detection. Second, if either party terminates the exchange in the middle, the advantage that party may gain over the other can be made arbitrarily small.

We transform the CEM problem into a symmetric exchange problem as follows: Sue enciphers her CEM with a cryptographic key and sends it in a signed commitment to Rob. Rob will then sign the received message to obtain the receipt. Rob enciphers the receipt with another cryptographic key and forms a signed commitment which he sends to Sue. Rob should also convince Sue that his signed commitment actually holds the valid receipt. Sue could, in polynomial-time, verify this if she knew the cryptographic key used by Rob. Rob uses a zero-knowledge interactive proof to convince Sue. When both signed

commitments are exchanged and Sue is convinced of the validity of Rob's signed commitment, the CEM problem has been transformed into a symmetric exchange problem. Both parties will then engage in a gradual and verifiable release of their respective cryptographic keys. One party goes first and both take tuns in releasing their cryptographic key commitments one bit at a time. By releasing a fraction of a bit or multiple bits, the granularity of the protocol could be adjusted. Note that the commitment schemes may need to be adjusted if the chosen granularity is not a single bit.

Protocol

The S-CEM protocol presented here consists of four phases. Each phase is described in greater detail in the following sections.

Phase one

Figure 15 illustrates Sue's activity in phase one of the S-CEM protocol. Sue signs her e-mail and then uses a conventional or symmetric cryptosystem to encipher her signature with a randomly generated cryptographic key, `S-key`. Sue's commitment to `S-key` is an ordered pair of bit-commitments for every bit of `S-key`. Her signed commitment is formed by appending the commitment to the ciphertext and signing the resulting message. Sue's signed commitment is then sent to Rob.

Phase two

Once received, Rob verifies Sue's signature on her signed commitment. Note that since `cem` is enciphered, Rob cannot verify its validity. However, Rob's receipt will depend on the received `cem`. If Sue's signature on `S-signed-commitment` was valid, Rob signs `S-signed-commitment` to form his receipt. Note that Rob can refuse to participate in the exchange (and thus refuse the mail) at this point. While we cannot force Rob to respond, we would like to detect whether he sends a valid receipt or not. Rob enciphers the receipt with a randomly generated key, `R-key`. Rob then forms a signed commitment and sends it to Sue. Note that since receipt is enciphered, Sue cannot verify its validity by simply looking at `R-signed-commitment`. Rob is therefore required to prove to Sue that the message sent actually contains a valid receipt. This can be done using a zero-knowledge interactive proof. We further elaborate on this point in the next section.

S	cem	:=	Sign(e-mail,S-secret-key)	15.1
S	S-key	:=	Random-Key-Generator()	15.2
S	ciphertext	:=	Encipher(cem,S-key)	15.3
S	commitment	:=	$\{\text{Bit-Commitment}(\text{S-key}_i,\text{S-r-i})\}_{i=1}^{\lvert\text{S-key}\rvert}$	15.4
S	message	:=	<ciphertext,commitment>	15.5
S	S-signed-commitment	:=	Sign(message,S-secret-key)	15.6
S-->R	S-signed-commitment			15.7

Figure 15. Phase one of an S-CEM protocol. Sender submits a signed commitment to recipient.

R	<S-ciphertext,S-commitment>:=Sign^{-1}(S-signed-commitment,S-public-key)			16.1
R	receipt	:=	Sign(S-signed-commitment,R-secret-key)	16.2
R	R-key	:=	Random-Key-Generator()	16.3
R	ciphertext	:=	Encipher(receipt,R-key)	16.4
R	commitment	:=	$\{\text{Bit-Commitment}(\text{R-key}_i,\text{R-r-i})\}_{i=1}^{\lvert\text{R-key}\rvert}$	16.5
R	R-signed-commitment	:=	Sign(<ciphertext,commitment>,R-secret-key)	16.6
R-->S	R-signed-commitment			16.7

Figure 16. Phase two of an S-CEM protocol. Recipient submits signed commitment to sender.

Phase three (sketch)

Rob must convince Sue that his signed commitment contains a valid receipt. As previously claimed, Rob can do this using a ZKIP. Rob acts as a prover and Sue acts as the verifier in an interactive dialogue. At the end of the dialogue, Sue is convinced (to a degree of certainty arbitrarily close to 1) that Rob's claim is valid. Using ZKIP, Sue cannot obtain any additional information on the signed commitment. In particular, Sue is no better off deriving the receipt from the signed commitment than if she had originally assumed Rob's claim to be true. In other words, Sue cannot cheat by trying to obtain additional information during the ZKIP. If Rob's signed commitment does not include a valid receipt, Rob should not be able to convince Sue otherwise.

So far we have claimed that such a ZKIP does exist. We will more formally prove our claim below. In order to do so, however, we need some common definitions and terminology. We use a formal-language framework and use the terminology presented in [8]. We define a language associated with signed commitments and show that this language is in the complexity class NP. Using a well known result from [14] stating that all languages in NP have ZKIPs, we then conclude that there exists a ZKIP for Rob with which he can convince Sue of the validity of his signed commitment.

Definition 2 - The SIGNED-COMMITMENT Language

Given the user identity, U, a message, m, and a signed commitment, sc, the problem is to find an ordered set of |k| random numbers, U-r-1 through U-r-|k| used to compute the commitment in sc. We assume that U's public key component is derivable from the user identity, U. Once k is extracted from the bit-commitments, it can decipher the ciphertext and result in U's signature of m. U's signature of m can easily be verified by applying the $Sign^{-1}$ function and using U's public key, U-public-key. The related decision problem can be termed: "Is there a sequence of |k| random numbers for which sc is a valid signed commitment of message m by U?". As a formal language, we define:

SIGNED-COMMITMENT = {<sc, m, U> $\in$ $\{0,1\}^*$: sc is U's signed commitment of m}

Lemma 1 - SIGNED-COMMITMENT $\in$ NP

Proof: To show that *SIGNED-COMMITMENT* $\in$ NP, for an instance <sc, m, U> of the problem, we let the |k| random numbers used in generating sc (U-r-1 through U-r-|k|) be the certificate used by a polynomial-time verification algorithm, Poly-Time-Verify. Poly-Time-Verify can then be constructed as presented in Figure 17. Note that if any sub-function fails, the Poly-Time-Verify function will return the integer zero (value of FALSE). The function first verifies the outer signature (see 17.1). The key, k, is then obtained by revealing the bit-commitments. Notice that if any random number received is different from the one used to generate the bit-commitment, the $Bit\text{-}Commitment^{-1}$ function fails and the whole function will return FALSE (17.2). U's signature of m is then obtained by deciphering the ciphertext using k (17.3). By applying the $Sign^{-1}$ function to the inner signature (17.4), a message is obtained which is compared for equality to m (17.5). Note that Poly-Time-Verify returns TRUE (value 1) if and only if all lines previous to (17.5) are executed successfully and the message obtained in (17.4) matches m. This means that there is no certificate that can fool the verification function into returning TRUE for invalid instances <sc′, m, U>, where sc′ is an invalid signed commitment of m by U.

The above algorithm (written in a pseudo-language) can be performed straightforwardly in polynomial-time as every sub-function used also runs in polynomial-time. Therefore, *SIGNED-COMMITMENT* $\in$ NP. ❑

Lemma 2 - For all languages L $\in$ NP, there exists a ZKIP.

Proof: This is a well known result. To show that there exists a ZKIP for every language in NP, it suffices to show some NP-complete language has a zero-knowledge interactive protocol. Using the method of polynomial-time reduction, we can then reduce all problems in NP into the NP-complete language chosen. There are several papers outlining a ZKIP for NP-complete languages in the literature. One example is the ZKIP for Graph 3-Colorability (assuming secure encryption exists) which was presented by Goldreich, Micali, and Wigderson in [14]. ❑

Theorem 1 - There exists a ZKIP for SIGNED-COMMITMENT

Proof: Proof follows immediately from Lemma 1, and Lemma 2. ❑

Phase four

At this point both parties are ready to engage in the symmetric exchange of information to reveal their respective key commitments (see Figure 18). Sue needs to send the |S-key| random numbers, S-r-1 through S-r-|S-key|, while Rob needs to send the |R-key| random numbers, R-r-1 through R-r-|R-key|. Note that |S-key| equals |R-key|. The gradual and verifiable exchange of the information could be achieved by both parties taking turns in the following iterative procedure. The protocol presented here assumes that Sue starts first. Rob follows

```
Boolean Poly-Time-Verify(<sc,m,U>,{U-r-i}_{i=1}^{|K|})
{
   <ciphertext,{Bit-Commitment(k_i,U-r-i)}_{i=1}^{|k|}>:= Sign^-1(sc,U-public-key)        17.1
   If the Sign^-1 function fails, return FALSE
   For (i=1 to |k|) Do  {                                                               17.2
         k_i:=Bit-Commitment^-1(Bit-Commitment(k_i,U-r-i),U-r-i)
         If the Bit-Commitment^-1 function fails, return FALSE
   }
   signature:=Decipher(ciphertext,k)                                                    17.3
   If the Decipher function fails, return FALSE
   message:=Sign^-1(signature,U-public-key)                                             17.4
   If the Sign^-1 function fails, return FALSE
   Compare(m,message)                                                                   17.5
   If NOT equal
         return FALSE
   else
         return TRUE
}
```

Figure 17. Polynomial time verification algorithm for SIGNED-COMMITMENT.

and executes symmetrically similar steps as Sue (presented in Figure 18). If either party cheats by sending invalid information, they would get caught (18.6). The parties abort the protocol if either is caught cheating. The party who cheated will have at most one bit more than the other. The cheater has therefore at most only a 2:1 computational advantage in finding the cryptographic key of the other party (assuming that the partial information obtained so far could be used to obtain the key).

Early stopping procedure

In what follows, Sue and Rob can act as both `Y` or `U`. If party `Y` cheats by sending a wrong random number, the other party, `U`, will abort the protocol. If only the last bit-commitment of `Y` was not revealed, `U` will guess the value of the last bit, $\texttt{Y-key}_{|\texttt{Y-key}|}$. On the other hand, if more bits of `Y-key` were missing, the only way `U` can obtain `Y-key` (other than by guessing) is to use some algorithm. The algorithm will use the partial information already obtained on `Y-key` and `Y`'s signed commitment, `Y-signed-commitment`. We assume that both parties

```
For (i=1 to |S-key|) Do                                                          18.1
{
S-->R      Sign(S-r-i,S-secret-key)                                              18.2
R-->S      Sign(R-r-i,R-secret-key)                                              18.3
S          R-r-i:=Sign^-1(Sign(R-r-i,R-secret-key),R-public-key)                 18.4
S          R-key_i:=Bit-Commitment^-1(Bit-Commitment(R-key_i,R-r-i),R-r-i)        18.5

If the Bit-Commitment^-1 function fails, then STOP (R is cheating)               18.6
}
If had to terminate early, then execute the early stopping procedure.
```

Figure 18. Phase four of an S-CEM protocol. Symmetric exchange to reveal (Sue's) commitments.

have the same knowledge of algorithms (Assumption I). Therefore as noted above, Y has at most only a fixed advantage over U. Therefore, our protocol is fair.

Assumptions unique to S-CEM protocols

In addition to the common assumptions enumerated earlier, S-CEM protocols have the following assumptions which follow from Assumption V and are stated here as corollaries.

D - Secure bit-commitment schemes exist

S-CEM protocols rely on the existence of a secure bit-commitment scheme as described earlier. The bit-commitment will allow for verifiable release of committed bits.

E - Unconditionally secure (for the verifier) ZKIPs with bounded execution time and bounded number of messages exchanged exist

The ZKIP we use should also be unconditionally secure for the verifier. This means that except perhaps for a negligible probability, the prover (Rob), no matter how powerful, should not be able to convince the verifier (Sue) of any false claim. The verifier must not learn any information beyond Rob's claim. We also require the ZKIP to complete with bounded number of messages exchanged and with finite execution time. This last condition is required for a practical implementation of the protocol.

Discussion

We informally discuss some issues of relevance in this section. In particular we discuss some aspects of the S-CEM protocol which was presented in this paper. We will also attempt to justify our approach.

Implementing the S-CEM protocol

The S-CEM protocol is to be performed while both parties are present on-line. This may limit the applicability of the S-CEM protocol for regular e-mail. On the other hand, interactive applications such as the File Transfer Protocol (FTP) [21] or the Post Office Protocol (POP) [26] can benefit from the on-line interactions in our S-CEM protocols. Of course other on-line applications which are based on the client-server model can also benefit from S-CEM.

Furthermore it is important to mention that while the presentation of our solution is theoretical, evidence suggests that practical implementations are possible. S-CEM implementations would require practical implementations of ZKIPs as well as great deal of tuning. Practical implementations of ZKIPs do exist. Protocols for several NP-complete problems are presented in the literature (see for example [4 and 6]). One can implement a zero-knowledge protocol for S-CEM using standard reduction techniques and existing zero-knowledge protocols for NP-complete problems. However, the resulting protocol would most likely be impractical (too many messages exchanged). Practical implementation of zero-knowledge protocols are available which require bounded number of messages exchanged during the protocol (see [10, 30, and 31]). Extrapolating from these existing protocols, we believe that practical implementation of ZKIP for S-CEM is possible.

Probability of undetected cheating

In S-CEM protocol, there are several cases to consider in which a party might attempt to cheat the other. In the first three phases, both parties may attempt to send each other an invalid signed commitment. Rob cannot cheat this way as he will most likely get caught during the ZKIP interactive dialogue (Corollary E). Sue also cannot cheat as the receipt is dependent on the content of the message received by Rob. If Sue sends junk, she receives a receipt indicating so. Unless Sue creates an invalid signed commitment for a bogus message which is encoded exactly the same as the valid signed commitment, Sue cannot benefit from cheating at this phase. To be successful, Sue must be able to do either of two things. She might try to find a message, m', not equal to the original message (`e-mail`, see 15.1) which when signed will result in the same signature as `Sign(e-mail,S-secret-key)`. Or, she might try to find a cryptographic key, $S\text{-}key'$, not equal to `S-key` which generates the same ciphertext as `Encipher(cem,S-key)`. In both possibilities Sue has a negligible chance of success (Corollaries C and B). The same is true for Rob. In other words, it is highly unlikely for Sue or Rob to be able to reliably achieve either of the above tasks in tractable time. In addition to the above, Sue (or Rob) may want to cheat by sending a bogus commitment string, one which she (he) could later reveal as a different key. Corollary D prevents this case except for an arbitrarily small probability. Hence, the chances that either party can cheat in the first three phases of the protocol can be made arbitrarily small. The exact probability depends on the actual algorithms used to implement the protocol. In the fourth phase of the S-CEM protocol, both parties engage in a gradual exchange of the bits to their respective cryptographic keys used in the protocol. This exchange is verifiable (Corollary D) using the $\texttt{Bit-Commitment}^{-1}$ function. Hence, the chances that either party can cheat can be made arbitrarily small. In the interaction in phase four, one party has some advantage over the other by not going first. The advantage is that the party has one extra

piece of information on the other's cryptographic key. This piece can be made arbitrarily small as shown by [28]. Hence, the advantage also can be made arbitrarily small. Therefore we argue that the protocol is *fair*.

APPLICATIONS

Electronic mail is one of the most popular services in any communication network. We believe that by enhancing the capabilities and security of electronic mail, CEM has the potential to make electronic mail an even more attractive service. Many applications that use *electronic messaging* can benefit from the use of CEM protocols. These include but are not limited to bank transactions, electronic funds transfer, trading of stocks, and any application that involves or can benefit from contractual agreements in digital form. Any service host providing services on the communication network can request a receipt for the use of its service. These receipts can then be used in the billing process.

Protocols for signing contracts can be accomplished using CEM. To sign a contract, *C*, Rob and Sue exchange two agreements as CEM messages. Sue sends Rob the CEM message: "Sue agrees to *C* if Sue receives Rob's agreement message. -- signed Sue". Rob sends a similar signed message to Sue interchanging every occurrence of Sue with Rob and vice versa. Note that the contract is referred to in each agreement message. Also note that both parties are verifiably committed to the contract only after both receive the receipt for their agreement message.

The B-CEM protocols presented above can be implemented in today's communications networks. Using PEM as a building block would provide for the required secure communications and allows B-CEM to take advantage of the growing interest and deployment opportunities associated with PEM. By enriching the features offered, B-CEM would help broaden the market base and applicability of PEM.

CONCLUSIONS

By enhancing the capabilities and security of electronic mail, CEM could make e-mail an even more popular service. Therefore, implementation of CEM deserves more attention. We believe that B-CEM protocols can be implemented with minimal modifications to existing e-mail systems and standard protocols. We further believe that the trusted postmaster agent could prove to be a useful building block for other new and exciting applications in the future. Actual implementations of the B-CEM protocols could use clever cryptographic techniques to improve the efficiency of the protocol. Some suggestions for efficient implementations have already been pointed out throughout the paper. Efficient implementations of S-CEM protocols are possible. However, their applicability is limited to on-line applications where both the sender and the recipient of the message are available on-line. High speed implementations of ZKIPs are necessary for the S-CEM family of protocols. We have not yet attempted to develop such a ZKIP. This is an interesting issue that deserves more attention. Further research, especially in the area of fast ZKIP techniques, is needed.

ACKNOWLEDGEMENTS

The original version of this paper [1] was written for the degree of Master of Science in Information Networking from the Information Networking Institute at Carnegie Mellon University. Steven Rudich provided invaluable input into the making of the original paper. Many papers in the literature, too numerous to list, were a true source of inspiration. The work of Manuel Blum [3] and Michael Rabin [22] on certified electronic mail is of particular interest and was analyzed in the original version of this paper.

For J. D. Tygar, partial support for this research came from the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597. Additional partial support was provided under a Presidential Young Investigator Award, Contract No. CCR-8858087, and by matching funds from Motorola Inc. and TRW. Additional partial support was provided by a contract from the U. S. Postal Service. The second author gratefully acknowledges IBM for a generous equipment grant.

Alireza Bahreman and J.D. Tygar can be reached via e-mail at bahreman@bellcore.com and tygar@cs.cmu.edu respectively.

BIBLIOGRAPHY

1. A. Bahreman. "Certified Electronic Mail," *Masters Thesis*, Information Networking Institute, Carnegie-Mellon University, July 1992.
2. D. Balenson. "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers," Internet Request For Comment 1423, February 1993.

3. M. Blum. “How to exchange (secret) keys,” *ACM Transactions on Computer Systems*, 1(2):175-193, May 1983.
4. G. Brassard. “Modern Cryptology: A tutorial,” Volume 325 of *Lecture Notes in Computer Science*. Spring-Verlag, 1985.
5. G. Brassard, D. Chaum, AND C. Crepeau. “Minimum disclosure proofs of knowledge,” *Journal of Computer and System Sciences*, 37(2):156-189, October 1988.
6. G. Brassard, AND C. Crepeau. “Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for SAT and beyond,” *Proceeding of the 27th Annual Symposium on Foundations of Computer Science*, pages 188-195, IEEE Computer Society, 1986.
7. CCITT/ISO. “X.500: The directory - overview and concepts, models and services,” CCITT/ISO IS 9594.
8. T. Cormen, C. Leiserson, AND R. Rivest. *“Introduction to algorithms,”* The MIT Press, 1990.
9. I. Damgard. “On the existence of bit commitment schemes and zero-knowledge proofs,” G. Brassard, editor, *Advances in Cryptology*, volume 435 of *Lecture Notes in Computer Science*, pages 17-27. Spring-Verlag, 1985.
10. U. Fiege, A. Fiat, AND A. Shamir. “Zero knowledge proofs of identity,” *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 210-217, 1987.
11. S. Goldwasser, AND S. Micali. “Probabilistic encryption,” *Journal of Computer and Systems Sciences*, 28(2):270-299, April 1984.
12. S. Goldwasser, S. Micali, AND C. Rackoff. “Knowledge complexity of interactive proof systems,” *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 291-304, 1985.
13. S. Goldwasser, S. Micali, AND C. Rackoff. “Knowledge complexity of interactive proof systems,” *SIAM Journal on Computing*, 18(1):186-208, February 1989.
14. O. Goldreich, S. Micali, AND A. Wigderson. “Proofs that yield nothing but their validity and a methodology of cryptographic protocol design,” *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 174-187, IEEE Computer Society, 1986.
15. R. Impagliazzo, L. Levin, AND M. Luby. “Pseudo-random generation from one way functions,” *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 12-24, 1989.
16. R. Impagliazzo, AND S. Rudich. “Limits on the provable consequences of one-way functions,” *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 44-61, May 1989.
17. International Standards Organization, ISO. “Basic reference model for open system interconnection,” ISO 7498, February 1984.
18. B. Kaliski. “Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services,” Internet Request For Comment 1424, February 1993.
19. S. Kent. “Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management,” Internet Request For Comment 1422, February 1993.
20. J. Linn. “Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures,” Internet Request For Comment 1421, February 1993.
21. J. Postel, AND J. Reynolds. “File Transfer Protocol,” Internet Request For Comment 959, October 1985.
22. M. Rabin. “Transaction protection by beacons,” *Journal of Computer and System Sciences*, 27(2):256-267, October 1983.
23. R. Rivest. “The MD4 message digest algorithm,” Internet Request For Comment 1320, April 1992.
24. R. Rivest. “The MD5 message digest algorithm,” Internet Request For Comment 1321, April 1992.
25. J. Rompel. “One-way functions are necessary and sufficient for secure signatures,” *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 387-394, 1990.
26. M. Rose. “Post Office Protocol - Version 3,” Internet Request For Comments 1460, June 1993.
27. S. Rudich. “Limits on the provable consequences of one-way functions,” *Ph.D. Thesis*, Department of Computer Science, University of California at Berkeley, December 1988.
28. T. Tedrick. “Fair exchange of secrets,” G. Blackley, AND D. Chaum, editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 434-438. Springer-Verlag, 1985.
29. T. Tedrick. “How to exchange half a bit,” D. Chaum, editor, *Advances in Cryptology: Proceedings of Crypto 83*. Plenum Press, 1983.
30. D. Tygar, AND B. Yee. “Strongbox: A system for self-securing programs,” R. Rashid, editor, *CMU Computer Science: A 25th anniversary commemorative*. ACM Press, 1991.
31. D. Tygar, AND B. Yee. “Strongbox,” Eppinger, J., Mummert, L., AND Spector, A., editors, *Camelot and Avalon: A distributed transaction facility.* Morgan Kaufman Publishers Inc., 1991.