A Certified Email Protocol using Key Chains

J. Cederquist
SQIG-IT and IST, TULisbon, Portugal

M. Torabi Dashti
CWI, Amsterdam, The Netherlands

S. Mauw
University of Luxembourg, Luxembourg

***Abstract*— This paper introduces an asynchronous optimistic certified email protocol, with stateless recipients, that relies on key chains to considerably reduce the storage requirements of the trusted third party. The proposed protocol thereby outperforms the existing schemes that achieve strong fairness. The paper also discusses the revocation of compromised keys as well as practical considerations regarding the implementation of the protocol.**

I. Introduction

Alice wants to send an email to Bob. She wishes to receive an *evidence of receipt* when Bob receives (and is able to read) the email. Bob is willing to send back an evidence of receipt to Alice only if he receives an *evidence of origin* along with Alice's email. Certified email (CEM) protocols are to provide such services.

CEM protocols are instances of *fair exchange* protocols. Similar to fair exchange protocols, there are three general constructions for CEM, based on the degree of the involvement of trusted third parties (TTP). The first class, which chronologically precedes the other classes, are CEM protocols with no TTPs, see for instance [1] and [2]. These are based on gradual release of information and require exchanging many messages to "approximate" fair exchange (CEM protocols with no TTPs are theoretically impossible, see [3]). Moreover, they often assume that the participants have equal computational powers. Protocols of the second class need the TTP's intervention in each exchange. Notable examples of these protocols are [4], [5], [6], [7] and [8]. A drawback of these protocols is that the TTP easily becomes a communication bottleneck or a single target of attacks. The third class of CEM protocols, known as *optimistic* fair exchange protocols, require the TTP's intervention only if a failure (accidentally or maliciously) occurs. Therefore, honest parties that are willing to exchange their items (e.g. emails and receipts) can do so without involving any TTP.

Optimistic fair exchange protocols basically consist of three sub-protocols (see, e.g., [9]): *exchange protocol*, *recovery protocol* and *abort protocol*. In the exchange protocol, that does not involve the TTP, the agents first *commit* to exchange their items and then they actually exchange them. An agent A can run the recovery protocol if the opponent B has committed to exchange, but A has not received B's item, and vice versa. A participant typically aborts (cancels) an exchange if she does not receive the opponent's commitment to the exchange. Optimistic protocols require the communication channels to and from the TTP to be *resilient*, i.e. messages are delivered within an arbitrary but finite amount of time. This guarantees that, in case of a failure, protocol participants can eventually consult the TTP. As examples of optimistic CEM protocols we refer to [9], [10], [11] and [12]. CEM protocols have also been extended to deal with multi-party exchanges, e.g. [13], [14] and [15], and certified mailing lists [16].

Contributions In order to achieve *strong fairness* (see section II-B), asynchronous optimistic fair exchange protocols require stateful TTPs [17]. Therefore, the amount of information that has to be stored by the TTP (virtually for an indefinite amount of time) is an issue in these protocols. In this paper we introduce an asynchronous optimistic CEM protocol, with stateless recipients, that aims at reducing the TTP's storage requirements using key chains [18], while guaranteeing strong fairness. A key chain is a sequence of keys such that each key is derived by applying a one-way function to the previous key. In section II-D, we thoroughly describe how key chains are used in our proposed CEM protocol.

We further optimize the protocol by using a practical aspect of CEM exchanges: Once two participants have started exchanging emails, usually several emails are exchanged between them. This observation motivates our proposed CEM protocol which imposes an initialization overhead, but is more efficient for exchanging several emails. To the best of our knowledge, this has not been previously studied

Structure of the paper In section II we start by describing the assumptions underlying the proposed protocol and its design goals. The protocol is presented in the rest of section II. In section III we discuss the security of the proposed protocol and analyze how it achieves the design goals. Section IV concerns practical issues and implementation considerations of the protocol. Section V compares our protocol with existing schemes and also points out some possible future work. We conclude the paper in section VI.

II. The protocol

In section II-A we present the assumptions that our protocol is based on, along with some notations. The design goals are described in section II-B. The protocol is presented in two steps: First we give a high level description of a naïve design of the protocol in section II-C. By first describing this, we show the ideas and point out some problems. However, the naïve design is not efficient and in section II-D we explain an efficient version of the protocol in detail.

A. Notations and assumptions

Throughout the paper $\mathcal{M}$ denotes the content of the email being exchanged. To denote that an agent A sends message m to agent B we write $A \rightarrow B : m$. The concatenation of the messages m_1 and m_2 is denoted m_1, m_2.

1) Communication channels assumptions: We assume resilient communication channels between each participant and the TTP. The channels connecting non-trusted protocol participants are however under the control of attackers. Therefore, no time bound on delivering messages is assumed for these channels (e.g. messages can get lost).

2) Cryptographic notations and assumptions: We assume ideal cryptographic apparatus à la Dolev and Yao [19]: A message m encrypted with the symmetric key K is denoted $\{m\}_K$, from which m can only be extracted using K. We assume the existence of a public key infrastructure. The notations $pk(X)$ and $sk(X)$ represent the public and private keys of entity X, respectively. In asymmetric encryption we have $\{\{m\}_{sk(X)}\}_{pk(X)} = \{\{m\}_{pk(X)}\}_{sk(X)} = m$. Encrypting with a private key denotes signing and, for convenience, for $\{m\}_{sk(X)}$ we write $(m)_X$. We also assume access to one-way collision-resistant hash functions.

3) TTP assumptions: We assume that the TTP T maintains a persistent secure database, with entries of the form $\langle X, Y, Z, W\rangle$, where X and Y are participant identities, Z is a key and W contains

21st International Conference on
Advanced Information Networking and Applications Workshops (AINAW'07)
0-7695-2847-3/07 $20.00 © 2007 IEEE

a random number (see section II-D.1). Associated with each such entry, T stores a linked list, initially of length zero. Each element of such a linked lists is of the form $(i, status(i))$, where $i \in \mathbb{N}$ and $status(i) \in \{0,1\}^l \cup \{\mathtt{a}\}$, for a finite l, and a special flag $\mathtt{a}$ that denotes an aborted exchange. In $status$, T stores the status of resolved exchanges, i.e. whether they have been recovered or aborted (see sections II-D.3 and II-D.4).

4) Idempotency assumption: Exchanged items, i.e. emails and evidences, are assumed idempotent. Therefore, to be able to receive an email or receipt twice is not different from receiving it once (meaning that it is not considered as an attack).

B. Design goals

One of the most fundamental requirements for CEM protocols is *non-repudiation*. Non-repudiation guarantees that an agent cannot deny having sent or received a message, if it has actually done so in the course of the protocol. To achieve this, protocol participants usually collect evidences, evidence of origin (EOO) and evidence of receipt (EOR), which can later be presented to a judge.

The second requirement for CEM protocols is to satisfy the *fair exchange properties* [9]. Here we aim at strong fairness as is defined below (for definitions of different levels of fairness see [20]). Fair exchange consists of three properties:

- *Effectiveness* states that if A and B engage in the protocol and are willing to exchange emails for receipts, then the protocol will reach a state where B has received the email content $\mathcal{M}$ and EOO, and A has received EOR, and both A and B have *terminated*, i.e. they have no further pending operations to perform in that protocol round.
- *Fairness* states that when the protocol round terminates, if A has received EOR, then B has received both $\mathcal{M}$ and EOO, and if A has not received EOR, then B possesses neither $\mathcal{M}$ nor EOO.
- *Timeliness* means that an honest participant can unilaterally, or with the help of a TTP, terminate the protocol run. Moreover, after termination, the degree of fairness does not decrease for an honest participant, i.e. if he did not receive his evidence or email content before terminating, then it cannot be that the other participant gets her evidence or email content at some future time.

Confidentiality is another requirement for CEM protocols which states that the exchanged email content should not be revealed to anyone (including the TTP), except to the intended receiver. Our proposed protocol does not directly address confidentiality. However, if confidentiality is desired, the exchanged email content $\mathcal{M}$ can be substituted with the actual email content encrypted for the receiver using his public-key or a shared secret key.

C. The naïve protocol

In most existing non-repudiation and CEM protocols, the initiator uses a separate key to encrypt the email to be exchanged, for each single protocol round. If the exchange goes amiss, the parties can resort to a TTP, that will store the key along with some other information about the exchange, such as involved parties, a hash value of the email content and an exchange label (e.g. see [21]). This information is stored virtually for an indefinite amount of time (see section V for more discussions).

We aim at reducing the amount of information stored by the TTP using *key chains* (c.f. [18]). A chain of keys is a sequence of keys $K_0, \ldots, K_n$ such that $K_i := H(K_{i-1})$, for $i > 0$, where H is a publicly known one-way function. The key K_0 is called the chain *seed*.

The key chain is initiated by the initiator Alice who chooses a seed K_0 and shares it with the TTP. They moreover agree on the maximum number n of exchanges that Alice can perform using the chain. Afterward, Alice traverses the chain backwards and she uses K_{n-i} (for $i \leq n$) to encrypt the message in the i^{th} exchange. Provided that the hash function H is preimage resistant, i.e. given $H(x)$ it is hard to compute x, the key used in i^{th} exchange remains unknown to the receiver Bob unless he knows one of the K_j for $j \leq i$. However, since Alice traverses the chain backwards, the keys seem to be fresh and independent.

The i^{th} exchange starts with Alice sending an email to Bob, encrypted using K_{n-i}, a key that Bob does not know (yet). Bob commits to the exchange by acknowledging the reception of the encrypted email. Afterward Alice sends Bob the key and, finally, Bob also acknowledges the reception of the key. In this scenario the TTP interferes only when a party does not receive the message he or she expects. Intuitively, when a party can prove that the opponent has committed to the exchange, then the TTP provides that party with the encryption key along with some affidavits. When key chains are used, in order to produce the key used in any resolved exchange, the TTP only needs to store the seed. The storage required is thus reduced considerably.

This protocol is trivially not purely optimistic because Alice needs to set up a chain with the TTP. However, if the number n of exchanges is large enough, then the gained reduction in required storage space of the TTP will, in many practical applications, compensate the overhead of the initial setup phase. But, one single problem prevents this idea to be practical: It is costly to abort an exchange. This is because of the following situation: Assume that exchange number i is aborted. This means that K_{n-i} is not revealed to Bob, but he gets hold of the encrypted email. This can happen for instance when Alice sends the encrypted email to Bob, but afterward changes her mind and aborts the exchange, for example because Bob is slow in replying. Now if the protocol proceeds and the $(i+1)^{th}$ session terminates successfully, then Bob learns $K_{n-(i+1)}$. Because of the way the chain is constructed, Bob can easily find out K_{n-i} (by computing $H(K_{n-(i+1)})$) and decrypt the email content of the aborted session at will. Fairness is thus violated. Therefore, if an exchange is aborted, Alice needs to abandon using the rest of the chain altogether and set up a new key chain with the TTP. This can potentially impose a huge efficiency penalty on the protocol. The next section describes a way to circumvent this problem.

D. The main protocol

Here we describe our asynchronous optimistic fair CEM protocol. This protocol uses keys in a key chain for encrypting emails that are exchanged. Once this chain has been initialized, emails can be encrypted and exchanged, each time with a new key. The protocol also provides a way for the initiator to revoke an entire key chain.

Each exchange (or attempt to exchange) that uses the optimistic protocol (and possibly also the recovery and abort protocols) is called a *protocol round*. An initialization phase followed by a number of protocol rounds is called a *protocol session*. After the initialization phase, the initiator can send emails to the responder. Each protocol session belongs to one unique initiator-responder pair. However, the protocol naturally allows concurrent sessions. An agent can thus be involved in different sessions with different opponents at the same time.

1) Initialization: First, the initiator A chooses a fresh random key K_0, the seed of the key chain. For $i \geq 0$, let $K_{i+1} := H(K_i)$ and $K'_i := H'(K_i)$, where H' is a publicly known secure hash function, while H can be an ordinary hash function. Moreover, we require that H and H' do not commute, i.e. for any x, $H'(x)$ does not reveal any information about $H'(H(x))$. The chain $K'_0, K'_1, \ldots$ of keys is used for encrypting the emails which are to be exchanged. Clearly any K'_i can be calculated from K_0 using H and H' (see figure 1). Agent A subsequently sends the seed K_0 and the identity of the potential responder (of the session) B to the TTP T:

$$\begin{array}{lll} 1. & A \to T: & \{A, B, K_0, nc\}_{pk(T)}, (A, B, K_0, nc)_A \\ 2. & T \to A: & sid, nc, \mathsf{cert}, (nc, \mathsf{cert})_T \end{array} \quad (1)$$

where the certificate $\mathsf{cert} := (A, B, sid)_T$ and nc is a fresh nonce chosen by A, and sid is a unique session identifier chosen by T.

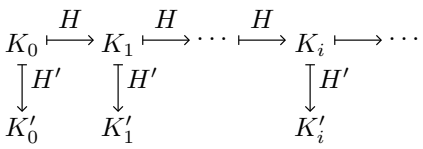

Fig. 1. Double key chain

T stores entries of the form $\langle initiator, responder, key, sid\rangle$, where key is the seed chosen by A. The key chain rooted at key can be used for sending CEMs from $initiator$ to $responder$. When T receives the first message, T checks the signature, and if it is valid then T looks for the entry $\langle A, B, K_0, *\rangle$ in its database. If $\langle A, B, K_0, *\rangle$ is not already present in its database, then T chooses a fresh session identifier sid and adds $\langle A, B, K_0, sid\rangle$ to the database. The TTP then sends back a confirmation of that it approves this session. If $\langle A, B, K_0, *\rangle$ already exists in the database, T ignores the request (and sends an error message to A).

2) Exchange protocol: Each protocol round has an order number i, which initially is 0, and after each round the initiator A increments it. The i^{th} protocol round is as follows

$$\begin{array}{lll} 1. & A \to B: & A, B, T, i, sid, h(K'_i), \{\mathcal{M}\}_{K'_i}, EOO_M, \mathsf{cert} \\ 2. & B \to A: & EOR_M \\ 3. & A \to B: & K'_i \\ 4. & B \to A: & EOR_{K'} \end{array} \quad (2)$$

where

- $EOO_M := (B, T, h(K'_i), h(\{\mathcal{M}\}_{K'_i}), i, sid)_A$
- $EOR_M := (EOO_M)_B$
- $EOR_{K'} := (K'_i, h(\{\mathcal{M}\}_{K'_i}))_B$

Here h is a secure hash function, possibly equal to H'. In the first message, A sends the encrypted email content $\{\mathcal{M}\}_{K'_i}$, the hash value of the encryption key $h(K'_i)$ and the session certificate $(A, B, sid)_T$. The responder B checks the correctness of the message and commits itself to receive the email by sending message 2, if he trusts T. Then A sends the key K'_i. Agent B checks that this key matches the hash value of the key that he received in message 1. Finally, if the key is correct, B sends a confirmation of having received the key. The number i is only used implicitly by B when it resolves the protocol round.

3) Recovery protocol: The initiator A may run the recovery protocol after having received message 2 in the exchange protocol, by presenting EOR_M to the TTP. This shows that A has actually sent EOO_M to B, ensuring that B is also able to receive a recovery token for that exchange. Agent A typically runs the recovery protocol to complete the EOR (as defined in section II-D.5 below), if it does not receive message 4 in the exchange protocol. The responder B may run the recovery protocol after receiving message 1 in the exchange protocol, in order to get the encryption key. The recovery protocol initiated by $P \in \{A, B\}$ starts with the following message:

$$1^r. \quad P \to T: \ f_r, A, B, h(K'_i), h(\{\mathcal{M}\}_{K'_i}), i, sid, EOR_M \quad (3)$$

where f_r is a flag used to identify the recovery request. On receiving this message, T performs the following tests:

- T checks if the signatures in the message are genuine and if its own identity is given as the designated TTP.
- T checks whether there is an entry in its database matching $\langle A, B, *, sid\rangle$.
- If the previous tests succeed, then the result of the query is a unique entry $\langle A, B, K_0, sid\rangle$ (see Initialization phase, section II-D.1). Subsequently T uses the retrieved K_0 to check whether $h(H'(H^i(K_0)))$ matches $h(K'_i)$ in the message.

If the results of all these tests are affirmative, then T checks whether round i has already been resolved or not. For each key chain (corresponding to one single entry $\langle A, B, K_0, sid\rangle$ in T's database) and each exchange $i \geq 0$ that is resolved at T, T stores whether that exchange has been recovered or aborted in a status variable $status(i)$ (c.f. section II-A.3). If $status(i)$ has not been initialized in T's database, it sets $status(i) := h(\{\mathcal{M}\}_{K'_i})$.[1] Then T proceeds as if $status(i)$ had already been set for the exchange, as is described below.

If $status(i)$ has already been initialized in the database, T sets $v := \perp$ if $status(i) = \mathsf{a}$, and sets $v := K'_i$ in case $status(i) = h(\{\mathcal{M}\}_{K'_i})$. Then T sends the following message and terminates this resolve.

$$2^r. \quad T \to P: \ v, (A, B, h(\{\mathcal{M}\}_{K'_i}), v, i, sid)_T \quad (4)$$

The message $(A, B, h(\{\mathcal{M}\}_{K'_i}), \perp, i, sid)_T$, where $\perp$ is a special flag that denotes an aborted exchange, serves as an abort token. When P receives this message, it can safely quit the protocol. The message $K'_i, (A, B, h(\{\mathcal{M}\}_{K'_i}), K'_i, i, sid)_T$ serves as a recovery token for P (see evidences in section II-D.5).

If $\neg(status(i) = h(\{\mathcal{M}\}_{K'_i}))$ or any of the tests mentioned above fails, then T ignores the recovery request and sends back the error message

$$2^r. \quad T \to P: \ f_e, (f_e, f_r, h(EOR_M))_T \quad (5)$$

where f_e is an error flag. This indicates a misbehavior, and based on this message, P can quit the protocol.

4) Abort protocol: The initiator A may abort an exchange at any stage, provided that the exchange has not been recovered already. Typically A aborts if it does not receive message 2 in the exchange protocol. On the other hand, the responder B can never explicitly request the TTP to abort an exchange that has been initiated by A. To abort an exchange, A sends the following message to T

$$1^a. \quad A \to T: \ f_a, A, B, h(\{M\}_{K'_i}), i, sid, \mathsf{abrt} \quad (6)$$

where $\mathsf{abrt} := (f_a, B, T, h(\{M\}_{K'_i}), i, sid)_A$ and f_a is a flag identifying the abort request.

On receiving this message, T checks A's signature on abrt and its own identity in the message, and it queries its database with $\langle A, B, *, sid\rangle$ only if they match. The result is a unique

[1]Note that if the hash function h produces hash values of l bits length, i.e. $h : \{0,1\}^* \to \{0,1\}^l$, then we have $status(i) \in \{\mathsf{a}\} \cup \{0,1\}^l$.

21st International Conference on
Advanced Information Networking and Applications Workshops (AINAW'07)
0-7695-2847-3/07 $20.00 © 2007 IEEE

$\langle A, B, K_0, sid\rangle$ (see Initialization phase). Then T checks whether session i has already been resolved, by checking whether $status(i)$ has been initialized in its database or not. If not, T sends back the following message:

$$2^a. \quad T \rightarrow A: \ (A, B, h(\{M\}_{K'_i}), \perp, i, sid)_T \tag{7}$$

This message serves as an abort token and when A receives it, A can safely quit the protocol. The TTP T then sets $status(i) := \mathsf{a}$ and terminates this resolve. Similarly, if $status(i)$ has already been set in T's database as $status(i) = \mathsf{a}$, then T sends the above message and terminates the resolve.

If $status(i)$ has already been set in T's database indicating a recovery, i.e. $\neg(status(i) = \mathsf{a})$, then T tests whether $status(i) = h(\{\mathcal{M}\}_{K'_i})$. If the test succeeds, T sends the following message and terminates this resolve:

$$2^a. \quad T \rightarrow A: \ (A, B, h(\{M\}_{K'_i}), K'_i, i, sid)_T \tag{8}$$

This message constitutes a recovery token for A, for this exchange.

If $\neg(status(i) = h(\{\mathcal{M}\}_{K'_i}))$ or if $\langle A, B, *, sid\rangle$ does not exist in T's database, or if any of the tests mentioned above fails, then T ignores the abort request and sends back an error message:

$$2^a. \quad T \rightarrow A: \ f_e, (f_e, \mathsf{abrt})_T \tag{9}$$

Note that an abort token does not necessarily mean that an exchange has not finished successfully, since A can abort an already completed exchange. An abort token merely indicates that T will never issue a recovery token (hence releasing the key) for that particular protocol round, uniquely identified with sid and i.

5) Evidences and dispute resolution: In case of a dispute, the parties can present evidences to an external judge. We note that each protocol round (of each protocol session) has an associated unique EOO and EOR.

The evidence of receipt EOR, desired by A, consists of

$$A, B, T, M, i, sid, K'_i, \mathsf{cert}, EOR_M, EOR_{K'},$$

if it is obtained by running the exchange protocol. If A uses the recovery or abort protocols, then the last element $EOR_{K'}$ is replaced by the recovery token $(A, B, h(\{\mathcal{M}\}_{K'_i}), K'_i, i, sid)_T$.

The evidence of origin EOO, desired by B, consists of

$$A, B, T, M, i, sid, K'_i, EOO_M$$

An external judge settles a dispute by simply checking whether the EOR or EOO presented by the disputing parties are genuine. We emphasize that abort tokens have no weight in these evidences. Therefore, having an abort token does not override or repudiate having a recovery token or any other evidence. The purpose of the abort protocol is solely to guarantee timeliness for the initiator.

6) Revoking compromised key chains: In practice it may happen that Alice's computer is compromised and the key chain seed is revealed to an attacker. In such situations, Alice might like to revoke the key chain she has set up with the TTP.[2] Therefore, the protocol allows Alice to ask the TTP, at any moment, to mark her key chain as *obsolete*:

$$A \rightarrow T: \ f_o, \mathsf{cert}, (f_o, \mathsf{cert})_A \tag{10}$$

Here f_o is a flag that denotes a request to mark the chain identified by cert as obsolete. Upon receiving this message, T checks A's signature and if cert is a genuine certificate from T to A, and (only if this is the case) marks the entry $\langle A, B, *, sid\rangle$ (which is unique) as obsolete. Marking an entry as obsolete means that T will not recover or abort protocol rounds connected to that entry any more. But it will behave as normal if it is queried about a protocol round for which a $status$ value has already been set. This behavior ensures that Alice cannot cheat Bob by first resolving an exchange and then marking the chain as obsolete (purporting that Bob would not be able to recover exchanges that use an obsolete chain).

We remark that revoking a key chain does not protect contents of emails that belong to previously aborted protocol rounds: If Bob compromises the key chain that Alice has used to send CEMs to him, then he can read the contents of emails from previous protocol rounds which were aborted.

[2]We assume that the attacker cannot get access to Alice's private key. Methods for revoking keys in PKIs have been extensively studied, e.g. see [22].

E. Discussions

1) Comparing the main protocol with the naïve protocol: Differently than in the protocol sketch in section II-C, in the main protocol (in section II-D) it is not needed to specify the maximum number n of exchanges. In the protocol sketch, n was used to specify the start point of the chain traversal as Alice needed to traverse the chain backwards. In the main protocol the whole chain is obscured using H'. So it is not needed to put any order on the key chain traversal. Hence, it is not needed to specify n.

2) Stateful vs. stateless recipients: Note that the receivers are stateless, i.e. they do not need to keep track of the protocol rounds or sessions that they have been involved in. However, they might want to store the evidences that they collect during the exchanges. It is nonetheless possible to require that the receivers store cert, i.e. $(A, B, sid)_T$, the first time they receive it (if they intend to continue accepting CEMs from the sender A). This certificate can subsequently be omitted from future protocol rounds between the sender and that particular receiver, hence reducing the communication costs.

III. Security analysis

In this section we justify the main protocol by informally showing that it achieves the design goals described in section II-B. A formal treatment of the security of the protocol is left for future work. We note that the structure of the protocol is similar to existing schemes, such as [12] and [21], except for the use of key chains. We thus focus the security analysis more on issues that are specific to the use of key chains in the protocol. For convenience, in the following discussions A and B denote the initiator and the responder parties, respectively.

- Non-repudiation: If A receives EOR (see section II-D.5), it can show that B has received both $\{\mathcal{M}\}_{K'}$ and K', and B is therefore able to extract $\mathcal{M}$. Moreover, B cannot deny that it is able to obtain $\mathcal{M}$ because of its signatures in EOR. Similarly, if B receives EOO, it can show that A has indeed sent K' and $\{\mathcal{M}\}_{K'}$, because of its signatures in EOO_M. Note that it is only A (or the TTP on behalf of A) who is able to generate K'.
- Effectiveness: If A and B follow the exchange protocol, A receives EOR iff B receives $\mathcal{M}$ and EOO.
- Fairness: As mentioned above (effectiveness), if the exchange protocol terminates normally then A receives EOR iff B receives $\mathcal{M}$ and EOO. However, if the exchange protocol is (mistakenly or maliciously) interrupted, then A can only recover the protocol round by presenting EOR_M to the TTP, hence proving that B has indeed received EOO_M in the interrupted exchange protocol. In this case B can also recover and receive the encryption key used in that round. If B recovers the exchange after it has received EOO_M but without sending EOR_M to A,

then A can (and needs to) abort in order to receive a recovery token for that exchange.

It may seem as if it could be a problem that A can reuse a key K_i' that has been used in a previous (possibly recovered or aborted) protocol round, to initiate a new round. But, as we will show here, it is of A's own interest to never reuse keys. Since $EOR_{K'}$ contains $h(\{\mathcal{M}\}_{K'})$, A needs either to send K' in the exchange protocol or to recover the exchange in order to receive the $EOR_{K'}$ and complete the EOR. In the first case, the exchange protocol terminates successfully, leaving A and B in the same fair state. In the second case, there are three possibilities: (1) The key K_i has been used in a protocol round that terminated normally: In this case, the protocol executes normally, i.e. as if i is a fresh (not reused) index. (2) The key K_i has been used in a protocol round that was recovered: If B is honest, B will also recover, and when they recover, both A and B receive an error message (unless the email content is actually exactly the same as what has already been recovered). This leaves A and B in the same (fair) state. (3) The key K_i has been used in a protocol round that was aborted: A receives neither the recovery token nor the last message from B. Thus, A cannot collect an EOR, neither can B collect EOO.

Potentially A could abort a completed exchange and then start a new protocol round with the same email content and key. In the new round, if A fails to continue the exchange protocol after receiving EOR_M, B receives an abort token when it tries to resolve, since the TTP has actually stored that round as aborted. But, because of the idempotency assumption (see section II-A.4), A does not gain anything more than what it had before reusing the key and the email content. Moreover, the abort token does not override the EOO that B has already collected. Thus the level of fairness achieved by B is not decreased.

We finally remark that if A maliciously uses a key that has been revealed to B earlier, then B can easily (by violating the protocol) obtain an EOO without sending message 2 in the exchange protocol.

- Timeliness: The agents A and B terminate a protocol round either by completing the exchange protocol, or by executing the recovery or abort protocol. Agent A can run the abort protocol at any time. It can also run the recovery protocol after it has received the second message in the exchange protocol. Agent B can recover the protocol after it has received the first message in the exchange protocol. Here we omit the details, but termination is guaranteed by the fact that the channels to the TTP are resilient. Note that termination of B depends on that it has the identity of the designated TTP signed by A in the first message of the exchange protocol (as it is in the main protocol). This is because A would otherwise be able to cheat B by resolving at a TTP which is not known (or trusted) to B, hence leaving B in an unfair situation.

  To show that the degree of fairness does not decrease for an honest participant after termination, it is enough to show that, if A (B) has terminated and not received EOR ($\mathcal{M}$ and EOO), then B (A) will not receive $\mathcal{M}$ or EOO (EOR). But, if A (B) has terminated and not received EOR ($\mathcal{M}$ and EOO), then we infer that the protocol round was aborted, and after an abortion B (A) cannot learn $\mathcal{M}$ or EOO (EOR).

IV. Implementation considerations

Hash chains can in practice be constructed using SHA-1 hash functions [23] as H'. A choice for H can be MD5 [24], so that H and H' do not commute. Other options for these would be to use various HMAC constructions [25] or secure block cipher encryption algorithms. In light of the recently discovered weaknesses of MD5 and the SHA family of hash functions (see e.g. [26]), it is important to notice that collision resistance is not of paramount importance to our protocol. There are two ways to counter collision attack. The first is to add redundancy to $\mathcal{M}$ (before encryption) in the first message of the exchange protocol. In this way a collision between the keys k and k' can be detected because the incorrect key yields an incorrect message after decryption. The second approach is to require that the TTP T determines the initial key K_0 instead of the initiator A. This solution will counter collision attacks, but will not help if the hash function is flawed with respect to second-preimage resistance.

A problem with hash chains is that when the length of a chain is increased, in practice the chance that a collision between that chain and another one occurs increases. A standard solution (e.g. see [27]) to this problem is to reduce the chance of collision by using the corresponding indexes when computing hash values: $K_i := H(K_{i-1}, i)$. For an in-depth discussion on constructing and implementing hash chains we refer the interested reader to [27].

We propose to use the AES encryption standard [28] for implementing symmetric key encryptions involved in the proposed protocol. Using hash functions to construct keys means that the length of the keys will be fixed, e.g. 160 bits if SHA-1 is used for H'. We note that currently AES with 128-bit keys is considered secure. Therefore, the result of the hash function can be truncated to 128 bits to fit into the AES standard.[3]

V. Related and future work

Related work In this paper we use hash chains of keys for repeated encryption in CEM protocols. The idea of using hash chains in security protocols can be traced back to [18], where they are used for repeated authentication. Hash chains have later on been used in authentication protocols, e.g. [30], [31], and key management systems, e.g. [32].

There is no general consensus in the literature on the requirements of CEM protocols: For example, in [5] and [8], it is not considered necessary for CEM protocols to provide EOO. In [33] it is argued that timeliness for the receiver is not required in CEM protocols. Conversely, in [11], timeliness for the sender is deemed unnecessary. In this paper we aim at strong fairness, which guarantees timeliness for both parties, and provides both EOO and EOR.

Below we study the efficiency and compare the TTP's storage requirements and the number of messages in the exchange protocol, between our proposed protocol and existing schemes. Existing schemes often require the TTP to store the key used for each single exchange along with the identities of the participants, the hash of the exchanged message and typically also a unique exchange identifier, e.g. see [21] for a review. In our protocol the TTP only needs to store one seed for each chain and the status of recovered or aborted rounds. When resolving, the TTP has to perform a few hash function computations in our protocol. However, these are in general very cheap. So, when exchanging multiple certified emails, our protocol outperforms the existing asynchronous optimistic CEM protocols (that provide strong fairness) in the amount of information stored by the TTP. Note that the protocols of [11], [33] and [34], which only need stateless TTPs, are not strongly fair, i.e. they do not guarantee timeliness for either sender or receiver. Other protocols

[3]The Rijndael algorithm, on which the AES is based, in fact allows using 160-bit keys, while the AES standard only supports 128, 192 and 256 bit key sizes [29].

with stateless TTPs, such as [6], are not optimistic, or rely on synchronous communication channels as in [35]. In fact, it has been shown in [17] that asynchronous optimistic contract signing (a cousin of CEM) with stateless TTPs cannot provide strong fairness.

Concerning the number of messages in the exchange protocol, according to [17], four messages is the least to achieve strong fairness for asynchronous optimistic contract signing protocols. Existing CEM protocols with only three messages in the optimistic phase are not strongly fair: The protocols of [11] do not guarantee timeliness of the sender. The protocols of [36], [37] (fixing a flaw in [36]), [38] (fixing a flaw in [36] and [37]), [15] and [39] (fixing a flaw in [15]) achieve fairness only under the rather unrealistic assumption that the cheater party collaborates with the suffered party and attends the court, in case of a dispute. In these protocols, since some of the collected evidences can conceptually be revoked based on other evidences (c.f. [40], [41]), only a weak notion of fairness is attainable.

Future work The design of the proposed CEM protocol certainly calls for formal verifications to ensure that by reducing the storage requirements of the TTP, no security flaws are introduced in the system. Whether the proposed protocol can be generalized to a multi-party setting is another question which needs further investigation.

VI. Conclusions

In this paper, we have introduced an asynchronous optimistic CEM protocol with stateless recipients. The protocol relies on key chains to reduce the storage requirements of the TTP, improving on existing schemes that achieve strong fairness. We have analyzed the protocol informally and showed that it guarantees non-repudiation, effectiveness, (strong) fairness and timeliness. We have also given a cryptographic justification to our usage of key chains, using standard cryptographic techniques.

Acknowledgment: We are grateful to Ricardo Corin for his contributions on earlier versions of the paper, in particular for the idea of using (single) hash chains to reduce the TTP's storage requirements, and to Ana Almeida Matos for proof reading the paper. The first author has been supported by the NWO project ACCOUNT and by the FEDER/FCT project QuantLog POCI/MAT/55796/2004.

References

[1] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Commun. ACM*, vol. 28, no. 6, pp. 637–647, jun 1985.

[2] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, "A fair protocol for signing contracts," *IEEE Trans. on Information Theory*, vol. 36, no. 1, pp. 40–46, jan 1990.

[3] S. Even and Y. Yacobi, "Relations among public key signature systems," Computer Science Dept., Technion, Haifa, Isreal, Tech. Rep. 175, March 1980.

[4] A. Bahreman and D. Tygar, "Certified electronic mail," in *Symp. on Network and Distributed Systems Security*. Internet Society, Feb. 1994, pp. 3–19.

[5] J. Zhou and D. Gollmann, "Certified electronic mail," in *ESORICS '96*. Springer, 1996, pp. 160–171.

[6] R. H. Deng, L. Gong, A. A. Lazar, and W. Wang, "Practical protocols for certified electronic mail," *J. Network Syst. Manage.*, vol. 4, no. 3, 1996.

[7] J. Riordan and B. Schneier, "A certified E-mail protocol," in *14th Annual Computer Security Applications Conference*. ACM, 1998.

[8] M. Abadi and N. Glew, "Certified email with a light on-line trusted third party: design and implementation," in *WWW '02*. ACM Press, 2002, pp. 387–395.

[9] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *IEEE Security and Privacy '98*. IEEE CS, May 1998, pp. 86–99.

[10] ——, "Optimistic fair exchange of digital signatures," in *Eurocrypt '98*, ser. LNCS, vol. 1403, 1998, pp. 591–606.

[11] S. Micali, "Simple and fast optimistic protocols for fair electronic exchange," in *PODC '03*. ACM Press, 2003, pp. 12–19.

[12] J. G. Cederquist, R. Corin, and M. Torabi Dashti, "On the quest for impartiality: Design and analysis of a fair non-repudiation protocol." in *ICICS '05*, ser. LNCS, vol. 3783. Springer, 2005, pp. 27–39.

[13] N. Asokan, M. Schunter, and M. Waidner, "Optimistic protocols for multi-party fair exchange," IBM Research, Research Report RZ 2892 (# 90840), dec 1996.

[14] S. Kremer and O. Markowitch, "A multi-party non-repudiation protocol," in *ISC '00*, ser. IFIP. Kluwer Academic, Aug. 2000, pp. 271–280.

[15] J. L. Ferrer-Gomila, M. Payeras-Capellà, and L. H. i Rotger, "A realistic protocol for multi-party certified electronic mail." in *ISC '02*, ser. LNCS, vol. 2433. Springer, 2002, pp. 210–219.

[16] H. Khurana and H.-S. Hahm, "Certified mailing lists," in *AsiaCCS '06*. ACM Press, 2006, pp. 46–58.

[17] B. Pfitzmann, M. Schunter, and M. Waidner, "Optimal efficiency of optimistic contract signing," in *PODC '98*. ACM Press, 1998, pp. 113–122.

[18] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[19] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. on Information Theory*, vol. IT-29, no. 2, pp. 198–208, 1983.

[20] H. Pagnia, H. Vogt, and F. C. Gärtner, "Fair exchange," *The Computer Journal*, vol. 46, no. 1, pp. 55–7, 2003.

[21] S. Gürgens, C. Rudolph, and H. Vogt, "On the security of fair non-repudiation protocols." *Int. J. Inf. Sec.*, vol. 4, no. 4, pp. 253–262, 2005.

[22] D. Boneh, X. Ding, G. Tsudik, and M. Wong, "A method for fast revocation of public key certificates and security capabilities," in *10th Usenix Security Symposium*, 2001, pp. 297–308.

[23] D. Eastlake and P. Jones, "US secure hash algorithm 1 (SHA1)," 2001, RFC 3174.

[24] R. Rivest, "The MD5 message-digest algorithm," 1992, RFC 1321.

[25] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *CRYPTO '96*. Springer, 1996, pp. 1–15.

[26] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *CRYPTO '05*. Springer, 2005, pp. 17–36.

[27] Y.-C. Hu, M. Jakobsson, and A. Perrig, "Efficient constructions for one-way hash chains." in *ACNS '05*, ser. LNCS, vol. 3531, 2005, pp. 423–441.

[28] NIST, "Advanced encryption standard (AES)," 2001, FIPS PUB 197.

[29] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[30] R. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. Needham, "A new family of authentication protocols," *SIGOPS Oper. Syst. Rev.*, vol. 32, no. 4, pp. 9–20, 1998.

[31] A. Perrig, J. D. Tygar, D. Song, and R. Canetti, "Efficient authentication and signing of multicast streams over lossy channels," in *IEEE SP '00*. IEEE CS, 2000, p. 56.

[32] J. Daemen, "Management of secret keys: Dynamic key handling." in *State of the Art in Applied Cryptography*, ser. LNCS, vol. 1528. Springer, 1998, pp. 264–276.

[33] G. Ateniese, "Verifiable encryption of digital signatures and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 1–20, 2004.

[34] A. Nenadic, N. Zhang, and S. Barton, "Fair certified e-mail delivery," in *SAC '04*. ACM Press, 2004, pp. 391–396.

[35] P. D. Ezhilchelvan and S. K. Shrivastava, "A family of trusted third party based fair-exchange protocols," *IEEE Trans. Dependable Secur. Comput.*, vol. 2, no. 4, pp. 273–286, 2005.

[36] J. L. Ferrer-Gomila, M. Payeras-Capell, and L. H. i Rotger, "An efficient protocol for certified electronic mail," in *ISW '00*. Springer, 2000, pp. 237–248.

[37] J. R. M. Monteiro and R. Dahab, "An attack on a protocol for certified delivery." in *ISC '02*, ser. LNCS, vol. 2433. Springer, 2002, pp. 428–436.

[38] G. Wang, F. Bao, and J. Zhou, "On the security of a certified e-mail scheme." in *INDOCRYPT '04*, ser. LNCS, vol. 3348. Springer, 2004, pp. 48–60.

[39] J. Zhou, "On the security of a multi-party certified email protocol." in *ICICS '04*, ser. LNCS, vol. 3269. Springer, 2004, pp. 40–52.

[40] S. Even, "A protocol for signing contracts," *SIGACT News*, vol. 15, no. 1, pp. 34–39, 1983.

[41] B. Pfitzmann, M. Schunter, and M. Waidner, "Optimal efficiency of optimistic contract signing," IBM Zürich Research Lab, Zürich, Tech. Rep. RZ 2994 (#93040), February 1998.