Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 274 (2011) 51–65
www.elsevier.com/locate/entcs

# Verification of A Key Chain Based TTP Transparent CEM Protocol

Zhiyuan Liu[a] Jun Pang[b] Chenyi Zhang[c]

[a] *University of Luxembourg, Luxembourg & Shandong University, China*
[b] *University of Luxembourg, Luxembourg*
[c] *University of Luxembourg, Luxembourg & University of New South Wales, Australia*

**Abstract**

In certified email (CEM) protocols, TTP transparency is an important security requirement which helps to avoid bad publicity as well as protecting individual users' privacy. Recently we have extended the CEM protocol of Cederquist et al. to satisfy TTP transparency. As a continuation, in this paper, we formally verify the security requirement in the exteded protocol. The properties of fairness, effectiveness and timeliness are checked in the model checker Mocha, and TTP transparency is analysed in the toolsets $\mu$CRL and CADP. The results confirm that our proposed extension achieves our design goals.

*Keywords:* Verification, fairness, timeliness, TTP transparency, CEM protocols, Mocha, $\mu$CRL

## 1 Introduction

Certified email (CEM) protocols, as an extension of regular email services, require that both senders and receivers be responsible for their roles in the email services. That means, as a protocol successfully runs to the end, neither the sender can deny the dispatch of the email, nor can the receiver deny the receipt. Such requirements are usually implemented by a *non-repudiable evidence of origin* (EOO) that is to be acquired by the receiver, and a *non-repudiable evidence of receipt* (EOR) that is to be acquired by the sender. Both the EOO and the EOR may serve as evidences in case of a dispute.

As a special class of fair exchange protocols [20], a CEM protocol is supposed to guarantee *fairness* with respect to non-repudiable evidences. Informally, at the end of a fair protocol run, either both parties acquire all the evidences, or no party gets an evidence. A *trusted third party* (TTP) can be introduced to take charge of the whole procedure and to provide undeniable records of submission (of the sender) and delivery (of the receiver). However in this way, a TTP may easily become a bottleneck, if she has to be involved in a large number of CEM services.


doi:10.1016/j.entcs.2011.07.006

A better solution, so called *optimistic* protocols [5], helps to release this burden from a TTP. In the optimistic protocols, a TTP is only required to be involved in case of unexpected events, such as a network failure or one party's misbehaviour, to restore fairness. If both the signer and the receiver behave correctly and there is no presence of significant network delays, a CEM protocol terminates successfully without intervention of the TTP. TTP *transparency* states that if a TTP has been contacted to help in a protocol, the resulting evidences will be the same as those obtained in the case where the TTP has not participated. In other words, by simply looking at the evidences, it is impossible to detect whether the TTP has been involved or not. Transparent TTPs are important and useful in practice, for instance, to avoid bad publicity. Moreover, this property also ensures privacy of the participants for asking for help from TTPs. In the context of CEM protocols, the use of a transparent TTP was first proposed by Micali [17], followed by a number of works, e.g., [16,18,19,21,12], in which different cryptographic schemes are used to achieve TTP transparency.

Recently, we have developed a CEM protocol with a transparent TTP [15], based on the protocol of Cederquist et al. [9] that applies key chains to reduce TTP's storage requirement. We achieve TTP transparency by adopting the verifiably encrypted signature scheme of [22]. We have shown that our extension is one of the most efficient CEM protocols satisfying *TTP transparency*, in addition to the other important properties such as *strong fairness*, *effectiveness*, and *timeliness*. The justifications to our claims are carried out on a rather informal level [15]. In this paper, we intend to put our analysis one step further, by incorporating formal verification techniques. The finite-state model checker Mocha [4] is used to verify the properties of fairness, timeliness and effectiveness, that are naturally interpreted in alternating-time temporal logic (ATL) formulas with game semantics [3]. The verification of properties expressed in ATL corresponds to the computation of winning strategies. Another toolset $\mu$CRL [7,6] is used for TTP transparency, which requires a comparison of observable traces in various situations. The $\mu$CRL toolset has the ability of generating state spaces that can be visualized and manipulated by the toolbox CADP [11] which acts as a back-end of $\mu$CRL.

*Structure of the paper.* We explain our proposed extension of the CEM protocol [9] and discuss its desired properties in Sect. 2. The two verification tools, Mocha and $\mu$CRL, are presented briefly in Sect. 3. In Sect. 4 we verify fairness, timeliness and effectiveness in Mocha with a focus on the modelling, and in Sect. 5 we verify TTP transparency in $\mu$CRL. Related work is discussed in Sect. 6. We conclude the paper in Sect. 7.

## 2 A Key Chain Based TTP Transparent CEM Protocol

Our protocol is developed on basis of the protocol [9], to support TTP transparency. Key chains are used to reduce TTP's storage requirement. Once a key chain is initialized between Alice and Bob, Alice can use any key within it to encrypt messages. Our approach requires the usage of a *verifiably encrypted signature scheme* to en-

code a receiver's commitment to receive the email.

For the sake of readability, we write Alice for the sender and Bob for the receiver. We assume the communication channels are *resilient*, in the sense that every message is guaranteed to reach its destination eventually. We write $\{M\}_k$ to denote a message $m$ encrypted with a symmetric key $k$, and $(M)_P$ to denote party $P$'s signature on message $M$. [1] We write $(M)_{B|T}$ for Bob's verifiably encrypted (partial) signature on $M$, by using the public key of TTP to encrypt Bob's signature on $M$. Everyone can verify that $(M)_{B|T}$ is authentic, but only TTP and Bob are able to extract the complete signature $(M)_B$ out of $(M)_{B|T}$.

### *2.1 The proposed protocol*

The structure of our protocol consists of an *exchange* sub-protocol, an *abort* sub-protocol and a *recover* sub-protocol. The exchange sub-protocol is executed by the communicating parties to deliver an email as well as exchanging undeniable evidences. The other sub-protocols are launched by a party to contact a TTP to deal with awry situations. Each exchange that uses the protocol is called a *protocol round*, and one initialisation phase followed by a number of protocol rounds is called a *protocol session*. Each protocol session belongs to a unique pair of communication parties.

*Key chain generation.* In optimistic CEM protocols, communicating parties will request TTP for help if the exchange process is disrupted. To achieve (strong) fairness, the TTP often needs to store sufficient amount of information, to have the ability to decrypt, retrieve or send out information for the protocol to finally reach a fair state. In most existing CEM protocols, the initiator uses either TTP's public key [18] or a separate key [21] to encrypt the email for each exchange. This first method normally requires asymmetric key operations, which are more expensive than symmetric key operations. The second method gives TTP burden of storing information of exchanges, such as involved parties, a hash value of email content and so on.

To reduce the TTP's burden of storing too much information, the protocol [9] uses *key chains*. A chain of keys is a sequence of keys $K'_0, \ldots, K'_n$, such that $K'_i := H(G^i(K_0))$ for each $i \geq 0$, where $K_0$ is the seed, $H : \kappa \to \kappa$ is a publicly known one-way collision-resistant hash function and $G : \kappa \to \kappa$ is a publicly known acyclic function ($\kappa$ is a key domain). $H$ and $G$ are non-commutative, i.e., given an $H(K_i)$ for which $K_i$ is unknown, it is infeasible to compute $H(G(K_i))$.

*Initialisation.* To initialise a session, the initiator Alice ($A$) sends the key chain seed $K_0$ and the identity of the potential responder Bob ($B$), together with a nonce $nc$ to the TTP ($T$). TTP will check whether there already exists an entry $\langle A, B, K_0, sid' \rangle$ in her database indicating whether the key chain has been established. If yes, TTP just ignores this request. Otherwise, TTP will choose a new session identity $sid$, and send the message $\mathsf{cert} := (A, B, sid)_T$ to Alice, and then store $\langle A, B, K_0, sid \rangle$ in her database.

[1] In practice a signature is always applied on a hashed value.

*Exchange sub-protocol.* The $i^{th}$ protocol round in a protocol session $sid$ is described below. The round number $i$ is initially 0 and can arbitrarily grow, Alice incrementing $i$ after each round. For convenience, we use $\mathsf{EOR}^{\frac{1}{2}}_M$ to denote $(\mathsf{EOO}_M)_{B|T}$.

$$1^{ex}.\ A \to B : A, B, T, i, sid, h(K'_i), \{M\}_{K'_i}, \mathsf{EOO}_M, \mathsf{cert}$$
$$2^{ex}.\ B \to A : \mathsf{EOR}^{\frac{1}{2}}_M$$
$$3^{ex}.\ A \to B : K'_i$$
$$4^{ex}.\ B \to A : \mathsf{EOR}_M$$

At first, Alice sends out message $1^{ex}$ to Bob. After receiving $\mathsf{EOO}_M$, Bob sends out his partial signature on $\mathsf{EOO}_M$ to show his commitment to receive the email. If Alice further sends Bob the key $K'_i$, Bob will deliver a full signature back to Alice as the final evidence of receipt.

*Abort sub-protocol.* Only Alice can abort, provided that the protocol has not yet been recovered. Typically, Alice aborts if she does not receive message $2^{ex}$. To abort an exchange, Alice sends TTP the following message:

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

where $f_a$ is a flag used to identify the abort request and $\mathsf{abrt}$ is Alice's signature on the abort request. After receiving this request, TTP checks several things such as the correctness of signatures, identities, entries for the key chain, and $status(i)$ to make decisions. If $status(i)$ has not been initialised, TTP will set it as aborted ($status(i) := \mathsf{a}$) and send back an abort token. If the current round has been recovered, TTP checks whether $status(i) = h(\{M\}_{K'_i})$. If yes, TTP will send back a recovery token. Otherwise, an error message of the form $(\mathsf{error}, (\mathsf{error}, \mathsf{abrt})_T)$ is sent back.

*Recovery sub-protocol.* Alice is allowed to launch the recovery sub-protocol provided she has sent out message $3^{ex}$, but has not received message $4^{ex}$. Similarly, Bob can launch the recovery sub-protocol if he has sent out message $2^{ex}$, but has not received message $3^{ex}$. The first message of the recovery sub-protocol for Alice is

$$1^{r}_A.\ A \to T : f_r, A, B, h(K'_i), h(\{M\}_{K'_i}), i, sid, \mathsf{EOR}^{\frac{1}{2}}_M, \mathsf{EOO}_M$$

where $f_r$ is a flag used to identify the recovery request. The first message of the recovery sub-protocol for Bob is

$$1^{r}_B.\ B \to T : f_r, A, B, h(K'_i), h(\{M\}_{K'_i}), i, sid, \mathsf{EOR}_M, \mathsf{EOO}_M$$

On receipt of a message for recovery, TTP needs to check (1) the correctness of (verifiably encrypted) signatures on $\mathsf{EOO}_M$ and $\mathsf{EOR}_M$ ($\mathsf{EOR}^{\frac{1}{2}}_M$), (2) the identity of TTP, and (3) whether there is an entry in its database matching $\langle A, B, \star, sid\rangle$. If all

the above checks succeed, TTP will retrieve $K_0$ and (4) check whether $h(H(G^i(K_0)))$ matches $h(K'_i)$. If yes, TTP will check $status(i)$ for round $i$.

- If $status(i)$ has not been initialised, TTP will set $status(i) := h(\{M\}_{K'_i})$. Whenever necessary TTP converts $\mathsf{EOR}^{\frac{1}{2}}_M$ into $\mathsf{EOR}_M$. After that, TTP sends out the following messages.

$$2^r.\ T \to B : K'_i, (K'_i)_T$$

$$3^r.\ T \to A : \mathsf{EOR}_M$$

- If $status(i) = h(\{M\}_{K'_i})$, then TTP performs step $2^r$ and step $3^r$ (again).
- If $status(i) = \mathsf{a}$, TTP sends out the abort token to the one that launched the protocol.

$$2^r.\quad T \to A(B) : \mathsf{abrt}, (\mathsf{abrt})_T$$

If any of the tests (1), (2), (3) and (4) fails, TTP ignores the recovery request and sends back an error message.

$$2^r.\quad T \to A(B) : \mathsf{error}, (\mathsf{error}, m^r)_T$$

where $m^r$ is the whole message received in step $1^r_A$ or $1^r_B$.

*Evidences and dispute resolution.* When a dispute occurs, both parties are required to provide evidences to an external judge. For each protocol round $i$, $\mathsf{EOO}$ (evidence of origin) desired by Bob consists of

$$A, B, T, M, i, sid, K'_i, \mathsf{EOO}_M.$$

$\mathsf{EOR}$ (evidence of receipt) desired by Alice consists of

$$A, B, T, M, i, sid, K'_i, \mathsf{cert}, \mathsf{EOR}_M.$$

### *2.2 Security requirements*

The following properties are claimed to be satisfied by the proposed protocol.

**Effectiveness.** If no error occurs then the protocol successfully runs till the end without any intervention from TTP.

**Timeliness**. Both Alice and Bob have the ability to eventually finish the protocol anywhere during the protocol execution. This is to prevent endless waiting of an honest party in case of unexpectancies.

**Fairness.** Honest Alice (Bob) will get her (his) evidences, provided that the other party gets the evidence from her (him).[2] The evidences can be used to convince an adjudicator that Bob has received the mail, in Alice's case, or that Alice is the true sender of the message, in Bob's case. A protocol satisfies fairness if every judgement on Bob's (Alice's) non-repudiation can be made solely and independently

[2] Note that only honest participants need to be protected.

from Alice's (Bob's) evidences, i.e., it does not necessarily involve TTP, nor the participation of Bob (Alice).

**TTP transparency**. The evidence each participant obtains is of the same format regardless of whether TTP is involved in the protocol execution or not.

# 3 A Brief Description of Mocha and $\mu$CRL

To formally analyse whether a security protocol achieves its design goals, first we have to specify the protocol in a formal language, and then express specifications for the desired properties. The model checker Mocha [4] allows specification of models with concurrent game structures, and expression of properties using ATL (Alternating-time Temporal Logic) [3] formulas with game semantics, which is suitable for checking properties such as *fairness, effectiveness* and *timeliness*. As to the analysis of TTP *transparency*, our main idea is to compare traces of getting evidences from different situations. [3] Therefore, a process algebraic language $\mu$CRL and its toolset [7,6] are used.

## *3.1 Mocha and ATL*

Mocha [4] is an interactive verification environment for the modular and hierarchical verification of heterogeneous systems. Its model framework is in the form of reactive modules. The states of a reactive module are determined by variables and are changed in a sequence of rounds. Mocha can check ATL formulas, which express properties naturally as winning strategies with game semantics. This is the main reason we choose Mocha as our model checker. Mocha provides a guarded command language to model the protocols, which uses the concurrent game structures as its formal semantics. The syntax and semantics of this language can be found in [4].

The temporal logic ATL is defined with respect to a finite set $\Pi$ of propositions and a finite set of players. An ATL formula is one of the following:

- $p$ for propositions $p \in \Pi$.
- $\neg\phi$ or $\phi_1 \vee \phi_2$, where $\phi$, $\phi_1$, and $\phi_2$ are ATL formulas.
- $\langle\langle A\rangle\rangle \bigcirc \phi$, $\langle\langle A\rangle\rangle \Box\phi$, or $\langle\langle A\rangle\rangle \phi_1 \mathcal{U} \phi_2$, where A$\subseteq \Sigma$ is a set of players, and $\phi$, $\phi_1$ and $\phi_2$ are ATL formulas.

ATL formulas are interpreted over the states of a concurrent game structure that has the same propositions and players [3]. The labeling of the states of a concurrent game structure with propositions is used to evaluate the atomic formulas of ATL. The logical connectives $\neg$ and $\vee$ have the standard meaning. Intuitively, the operator $\langle\langle A\rangle\rangle$ acts as a selective quantification over those paths that the agents in $A$ can enforce. The path quantifiers $\bigcirc$ (next), $\Box$ (globally) and $\mathcal{U}$ (until) carry their usual meanings as in the logic CTL, and $\Diamond\phi$ is defined as $true\,\mathcal{U}\phi$.

[3] This cannot be done with Mocha.

### 3.2 μCRL and CADP

μCRL is a language for specifying distributed systems and protocols in an algebraic style. A μCRL specification consists of two parts: one part specifies the data types, the other part specifies the processes.

The data part contains equational specifications; one can declare sorts and functions working upon these sorts, and describe the meaning of these functions by equations. Processes are represented by process terms. Process terms consist of action names and recursion variables with zero or more data parameters, combined with process-algebraic operators. Actions and recursion variables carry zero or more data parameters. Intuitively, an action can execute itself, after which it terminates successfully. There are two predefined actions: $\delta$ represents deadlock, $\tau$ the internal action. $\mathsf{p}.\mathsf{q}$ denotes sequential composition, it first executes $\mathsf{p}$ and then $\mathsf{q}$. $\mathsf{p}+\mathsf{q}$ denotes non-deterministic choice, meaning that it can behave as $\mathsf{p}$ or $\mathsf{q}$. Summation $\sum_{\mathsf{d}:\mathsf{D}} \mathsf{p}(\mathsf{d})$ provides the possibly infinite choice over a data type $\mathsf{D}$. The conditional construct $\mathsf{p} \lhd \mathsf{b} \rhd \mathsf{q}$, with $\mathsf{b}$ a boolean data term, behaves as $\mathsf{p}$ if $\mathsf{b}$ and as $\mathsf{q}$ if not $\mathsf{b}$. Parallel composition $\mathsf{p}\|\mathsf{q}$ interleaves the actions of $\mathsf{p}$ and $\mathsf{q}$; moreover, actions from $\mathsf{p}$ and $\mathsf{q}$ may synchronise into a communication action, if explicitly allowed by a predefined communication function. Two actions can only synchronise if their data parameters are the same, which means that communication can be used to capture data transfer from one process to another. If two actions are able to synchronise, then in general we only want these actions to occur in communication with each other, and not on their own. This can be enforced by the encapsulation operator $\partial_H(\mathsf{p})$, which renames all occurrences in $\mathsf{p}$ of actions from the set $H$ into $\delta$. Additionally, the hiding operator $\tau_I(\mathsf{p})$ turns all occurrences in $\mathsf{p}$ of actions from the set $I$ into $\tau$.

The μCRL tool set [7] is a collection of tools for analysing and manipulating μCRL specifications. The μCRL tool set, together with the CADP tool set [11], which acts as a back-end for the μCRL tool set, features visualisation, simulation, LTS generation and minimisation, model checking, theorem proving and state-bit hashing capabilities.

## 4 Verification of the Protocol in Mocha

We give a sketch of our modeling approach and discuss the built models for the extended CEM protocol. Detailed models and analysis can be found in [14].

### 4.1 Modeling the protocol in Mocha

At first, each participant is modelled as a player (in a game), with the description of its behaviours using the guarded command language of Mocha. Models for honest participants can be easily specified strictly in accordance with the protocol. As to the dishonest models, we mainly consider the dishonest participant's behaviours, since the security of CEM protocol can be hazarded by dishonest participants instead of outside intruders. Therefore, we build models for both honest and dishonest

participants. For each participant, we write $P_i$ and $P_iH$ to represent the dishonest and honest models, respectively. Intuitively, dishonest model $P_i$ allows the player to cheat, while $P_iH$ just follows the protocol honestly. Dishonest behaviours include sending messages derivable from his knowledge at any time, stopping at any time; therefore, a dishonest model may not stop at a point where its role in the protocol is required to stop.

Communication is modelled using shared variables. Evidences (EOO and EOR), key and emails are modelled as boolean variables which are initialised as false and updated by its sender. We model the action of sending out an evidence, or other messages as a guarded command in which the sender resets the corresponding variables as true. In the model for honest participant $P_iH$, the guard consists of all the conditions to be satisfied strictly according to the protocol, and the command consists of all the corresponding actions to be executed. However, for the dishonest $P_i$, the guard just consists of necessary messages to generate the message to be sent.

List. 1 gives the Mocha code describing the behaviours of honest Alice. At first, Alice can do idle actions after she initiates a protocol round by sending out $\mathsf{EOO}_M$. For honest Alice, she mainly performs two kinds of actions in the exchange sub-protocol, which includes sending evidence of origin and the key. They are described in *step* (1) and (2). *Step* (1) models the action of sending $\mathsf{EOO}_M$, in which we use boolean variables *hk* and *pa_eoo* to represent the hashed value of $K_i'$ and the message $(B, T, i, sid, h(K_i'), \{M\}_{K_i'})_A$ signed by Alice, respectively. Setting *hk* and *pa_eoo* to true means Alice has initiated a communication with Bob by sending out her $\mathsf{EOO}_M$. *Step* (2) says that if Alice has received the correct verifiably encrypted message, namely *pb_halfeorm* has become true, she can set *k* as true, which represents the action of sending out key $K_i'$. Except for the exchange sub-protocol, Alice is also able to initiate the abort protocol if she does not receive the verifiably encrypted signature *pb_halfeorm* from Bob. This abort request *A_abort_req* is described in *step* (4), in which the guard represents the requirements for asking for abort from TTP, and the commands represent the behaviour of contacting TTP for abort. Besides the abort sub-protocol, Alice can also initiate the recovery sub-protocol which is modelled in *step* (6). Recovery request is modelled as a boolean variable *A_recovery_req*, and it will be set to be true if the guard is satisfied, in which the *k* and *pb_halfeorm* are true while *pb_eorm* is false. Note that once honest Alice initiates a recovery or abort sub-protocol with TTP, she will not continue the exchange sub-protocol. This mechanism is realized by modeling a boolean variable *A_contacted_T*. Finally, Alice can stop if she receives final $\mathsf{EOR}_M$ from Bob (*step* (3)) or recovery token from TTP (*step* (7)). Abort token (*step* (5)) can also make Alice stop the protocol round. In a similar way, we can model the honest behaviours of Bob.

Listing 1: Extracted honest model of Alice for the extended CEM protocol

```
-- idle actin while not stopped
    [] ~pa_stop & pa_eoo ->
-- (1) Alice sends EOO to Bob
    [] ~pa_stop & ~A_contacted_T & ~pa_eoo
       -> pa_eoo':=true; hk':=true
-- (2) Alice sends out key while receiving half EOR_M
```

```
    [] ~pa_stop & ~A_contacted_T & pb_halferom & ~k
       -> k':=true
-- (3) Alice can stop when she receives Bob's EOR
    [] ~pb_stop & ~A_contacted_T & pb_eorm & ~pa_rece_eorm
       -> pa_rece_erom':=true
-- (4) Alice can send out abort request to TTP
       if she hasn't received half EOR_M from Bob
    [] ~pa_stop & ~A_contacted_T & pa_eoo & ~pb_halfeorm
       -> A_contacted_T':=true; A_abort_req':=true
-- (5) Alice stops after receiving abort token from TTP
    [] ~pa_stop & A_contacted_T & T_abort_send_A
       -> T_abort_token_A':=true; pa_stop':=true
-- (6) Alice can send recovery request
       while she possesses pb_halfeorm
    [] ~pa_stop & ~A_contacted_T & k & pb_halfeorm & ~pb_eorm
       -> A_contacted_T':=true; A_recovery_req':=true
-- (7) Alice stops after receiving recovery token from TTP
    [] ~pa_stop & T_recovery_send_A ->
       pa_rece_eorm' := true; pa_stop':=true
```

List. 2 describes the behaviours of dishonest Alice, her malicious behaviours are described as follows. At first Alice is allowed not only to idle, but also to stop and to quit the protocol at any time she wants. The behaviours of sending $\mathsf{EOO}_M$ and the key are specified in *step* (1) and (2). *Step* (1) models that Alice can send out her evidence of origin by setting variable *pa_eoo* to true at any time she wants, even if she has already contacted TTP and is supposed to stop. Together with *pa_eoo*, malicious Alice still has the choice of sending out correct hashed key *hk* or incorrect hashed key *hke*. Similarly, *step* (2) specifies that Alice can send out her key at any time she wants. If the variable $k$ is true, it means that the correct key has been sent out. Otherwise, it represents that Alice has not sent out any key or the key that has been sent out is wrong. Moreover, *step* (3) and (4) models that Alice can contact TTP for abort or recovery as long as she has received enough messages, but she does not set the *A_contact_T* as true. The last two steps describe the situations when Alice has received $\mathsf{EOR}_M$ or an abort token from TTP.

Listing 2: Extracted dishonest model of Alice for the extended CEM protocol

```
-- idle actin while not stopped
    [] ~pa_stop & pa_eoo ->
-- Alice can stop at any time
    [] ~pa_stop & pa_eoo -> pa_stop':=true
-- (1) Alice can send EOO at any time
  --send correct hashed key
    [] ~pa_stop & ~pa_eoo & ~hk & ~hke
       -> pa_eoo':=true; hk':=true
  --send incorrect hashed key
    [] ~pa_stop & ~pa_eoo & ~hk & ~hke
       -> pa_eoo':=true; hke':=true
-- (2) Alice can send out key at any time
    [] ~pa_stop & ~k -> k':=true
-- (3) Alice can send abort request
    [] ~pa_stop & pa_eoo -> A_abort_req':=true
-- (4) Alice can send recovery request
    [] ~pa_stop & pb_halfeorm -> A_reovery_req':=true
-- (5) Alice receives abort token
    [] ~pa_stop & T_abort_send_A -> T_abort_token_A':=true
-- (6) Alice receives recovery token
    [] ~pa_stop & T_recovery_send_A -> pa_rece_eorm':=true
```

In a similar way, we can model the dishonest behaviours of Bob.

List. 3 models the corresponding behaviours of TTP. TTP is a special player that has to be modelled in a particular way. It must be objective, and cannot act in collusion with protocol participants. We build the model for TTP that strictly

follow the protocol. For each protocol round, we use a variable $T\_stateAB$ to record the status of protocol. $T\_stateAB$ has three possible values, which are *abrt*, *recov* and *empty* representing aborted, recovered and empty states, respectively. After receiving recovery or abort request, TTP will behave according to the values of $T\_stateAB$. The first part describes how TTP deals with abort request from initiator Alice. TTP sends out abort token to both Alice and Bob if the status is *empty* or *abrt*, and the $T\_stateAB$ is also needed to be set as *abrt* if the original status is *empty*. However, if $T\_stateAB$ is *recov*, which means the corresponding round has already been recovered, then the corresponding $\mathsf{EOR}_M$ and key must be sent to Alice and Bob respectively. Part two and three models the behaviours of dealing with recovery requests from Alice and Bob. If the TTP receives a recovery request and its status is *empty* or *recov*, then the required evidences or key must be sent to Alice and Bob respectively. Otherwise, abort token will be sent out.

Listing 3: Extracted model of TTP for the extended CEM protocol

```
-- (1) If TTP receives abort request from Alice
    [] A_abort_req & (T_stateAB=abrt) & ~T_response_A
      -> T_abort_send_A':=true; T_abort_send_B':=true;
         T_response_A':=true
    [] A_abort_req & (T_stateAB=empty) & ~T_response_A
      -> T_abort_send_A':=true; T_abort_send_B':=true;
         T_response_A':=true; T_stateAB':=abrt
    [] A_abort_req & (T_stateAB=recov) & ~T_response_A
      -> T_recovery_send_A':=true; T_recovery_send_B':=true;
         T_response_A':=true
-- (2) If TTP receives recovery request from Alice
    [] A_recovery_req & (T_state=empty) & ~T_response_A ->
      -> T_stateAB':=recov; T_recovery_send_A':=true;
         T_recovery_send_B':=true; T_response_A':=true
    [] A_recovery_req & (T_state=recov) & ~T_response_A ->
      -> T_recovery_send_A':=true;T_recovery_send_B':=true;
         T_response_A':=true
    [] A_recovery_req & (T_state=abrt) & ~T_response_A ->
      -> T_abort_send_A':=true;T_abort_send_B':=true;
         T_response_A':=true
-- (3) If TTP receives recovery request from Bob
    [] B_recovery_req & (T_state=empty) & ~T_response_B ->
      -> T_stateAB':=recov; T_recovery_send_A':=true;
         T_recovery_send_B':=true; T_response_B':=true
    [] B_recovery_req & (T_state=recov) & ~T_response_B ->
      -> T_recovery_send_A':=true;T_recovery_send_B':=true;
         T_response_B':=true
    [] B_recovery_req & (T_state=abrt) & ~T_response_B ->
      -> T_abort_send_A:=true;T_abort_send_B':=true;
         T_response_B':=true
```

### *4.2 Expressing properties of the protocol in ATL*

Given a CEM protocol with just two participants Alice and Bob, the following expressions are suitable for honest participant even if the other is dishonest. Actually, we only care about fairness and timeliness for honest participant. As to effectiveness, it requires that both participants must behave honestly.

**Effectiveness.** If honest participants are willing to exchange emails for receipts, then the protocol will terminate in a state that Alice has obtained $\mathsf{EOR}$ and Bob has received $\mathsf{EOO}$ and $M$ without the involvement of TTP.

$$\mathit{effectiveness} \equiv (\langle\langle P_a H, P_b H\rangle\rangle \Diamond (\mathsf{EOO} \wedge M \wedge \mathsf{EOR}))$$

where $P_aH$ and $P_bH$ represent honest participants Alice and Bob, and EOR represents the evidence of receipt from receiver Bob. In addition, the EOO and $M$ represents the evidence of origin and the email content from Alice.

**Timeliness.** At any time, an honest participant has a strategy to stop the protocol and thus to prevent endless waiting. Timeliness for Alice and Bob is formulated as:

$$timelinessP_a \equiv \forall\Box\,(\langle\langle P_aH\rangle\rangle\Diamond\, P_a\_stop) \quad timelinessP_b \equiv \forall\Box\,(\langle\langle P_bH\rangle\rangle\Diamond\, P_b\_stop).$$

where $P_aH$ and $P_bH$ represent the honest Alice and Bob, and $P_a\_stop$ ($P_b\_stop$) represents that Alice (Bob) has already terminated the protocol.

**Fairness.** A protocol is fair for honest Alice $P_a$ if the following is satisfied: whenever Bob obtains $P_a$'s non-repudiation evidence of origin (EOO) and email content $M$, $P_aH$ has a strategy to obtain Bob's non-repudiable evidence of receipt (EOR). In ATL, fairness for honest Alice can be formulated as:

$$fairnessP_aH \equiv \forall\Box\,((\mathsf{EOO} \wedge M) \Rightarrow \langle\langle P_aH\rangle\rangle\Diamond\,(\mathsf{EOR})).$$

Similarly, fairness for Bob is formulated as below. If Alice obtains $P_bH$'s EOR, honest Bob $P_bH$ has a strategy to get Alice's EOR and email content $M$.

$$fairnessP_bH \equiv \forall\Box\,((\mathsf{EOR}) \Rightarrow \langle\langle P_bH\rangle\rangle\Diamond\,(\mathsf{EOO} \wedge M)).$$

### *4.3 Analysis*

We have built three Mocha models, $P_aH \parallel P_bH \parallel TTP$, $P_a \parallel P_bH \parallel TTP$, and $P_aH \parallel P_b \parallel TTP$, combining the aforementioned formulas, to verify fairness, timeliness and effectiveness of our CEM protocol. These properties were successfully checked in Mocha.

## 5 Verification of the Protocol in $\mu$CRL

In this section, we only give a sketch on how we model the protocol in $\mu$CRL, and focus on how to check TTP transparency of the protocol in $\mu$CRL. The detailed models and analysis can be found in [14].

### *5.1 Modeling the protocol in $\mu$CRL*

As stated in Sect. 5, each $\mu$CRL specification consists of two parts, which are abstract data type definitions and behavioural specifications for participants. Since the execution of protocol mainly depends on the exchange of messages, the contents of the data are not treated in details, instead the data type used and corresponding operations on it are captured. Therefore, we can simplify the complex cryptographic primitives, such as encryption, decryption and verifiably encryption of messages.

TTP transparency states that the final evidences do not reveal whether TTP has intervened in the protocol or not. The main idea of checking TTP transparency

is to compare traces obtained from three different models after hiding all unnecessary actions, such as messages between TTP and the users, as well as minimising the generated state space modulo weak trace equivalence. The three models are combinations of honest Alice and honest Bob, honest Alice and malicious Bob and TTP, and malicious Alice and honest Bob and TTP.

Participants are hooked up by communication channels. According to our assumption, the communications channels are resilient, in the sense that every message is guaranteed to reach its destination eventually. Therefore, by using the encapsulation and communication operators in $\mu$CRL, we are able enforce the actions of participants Alice, Bob and TTP to synchronise. Each participant is defined as a process. The communications between them are composed by actions of sending and receiving messages. The honest and dishonest behaviours of the participants resemble those in the Mocha models.

For instance, the behaviours of the initiator Alice are modelled in a process with a parameter key, which initiates the CEM protocol by sending evidence of origin EOO to receiver Bob. The action init_Alice(x,y,i,A,B) shows that Alice initiates a protocol round $i$ for delivering an email $y$ to Bob using a key $x$. Then after receiving the verifiably encrypted message from Bob, honest Alice will send out her key. If Bob's final reply EOR is correct, Alice will be sure that she has completed one email delivery and successfully obtained the evidence of receipt. Action evidence_Alice(x,y,i,halfeorm,eorm,A,B) reports that she has already obtained the evidence for protocol round $i$ which sends email $y$ with key $x$. The sketch of Alice's behaviour is described as follows.

$$\begin{aligned}\mathsf{Alice}(\mathsf{x}:\mathsf{Key}) = \textstyle\sum_{\mathsf{y}:\mathsf{Item}} \sum_{\mathsf{i}:\mathsf{Number}} \ & \mathsf{initSend}(\mathsf{A},\mathsf{eoo},\mathsf{B}).\mathsf{init\_A}(\mathsf{A},\mathsf{y},\mathsf{x},\mathsf{i},\mathsf{B}) \\ & \mathsf{recv}(\mathsf{B},\mathsf{halfeorm},\mathsf{A}).\mathsf{send}(\mathsf{A},\mathsf{k},\mathsf{B}). \\ & \mathsf{recv}(\mathsf{B},\mathsf{eorm},\mathsf{A}).\mathsf{evidence\_A}(\mathsf{A},\mathsf{y},\mathsf{x},\mathsf{i},\mathsf{eorm},\mathsf{B})\end{aligned}$$

where eoo represents the the first message $1^{ex}$ for protocol round $i$. The halfeorm and eorm represents Bob's verifiably encrypted signature and final signature. We need to extend the above process when taking TTP into account to cover when Alice can contact TTP and receive replies from TTP. Similarly, honest Bod, dishonest Alice, dishonest Bob, and TTP can be modelled in $\mu$CRL as well, by specifying their behaviours as discussed before.

### *5.2 Analysis*

Our way to check TTP transparency is by comparing traces of getting evidences between system of only honest participants and systems containing dishonest participants. After hiding some actions and reducing the model, we obtain a trace from the honest system that is depicted in Fig. 1(a), which shows the situation of getting evidences without TTP. Fig. 1(b) describes traces obtained from the model containing honest Alice, dishonest Bob, and TTP. We can find that Fig. 1(b) has one more trace. Evidences for both traces are of the same form, but the sequence of getting

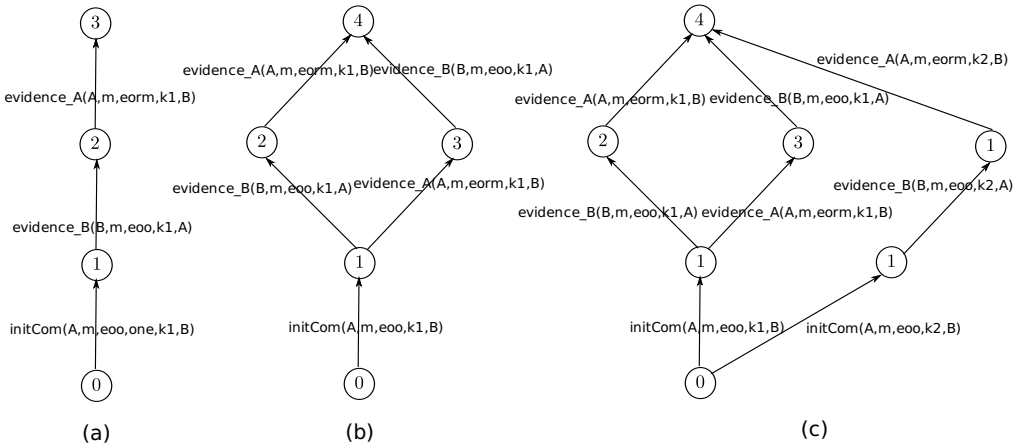


Fig. 1. The obtained traces.

them are different. However, this difference does not affect the correctness of TTP transparency. When checking the evidences possessed Bob and Alice, the only thing that matters is the content of the evidences, and the number of transitions (which might reflect the execution time) is irrelevant due to the asynchrony of the protocol model. Fig. 1(c) depicts the traces obtained from the model containing dishonest Alice, honest Bob and TTP. We can find that this figure has one more trace than Fig. 1(b). This extra trace describes Alice's malicious behaviours of using the key ($k2$) that does not match the protocol round ($i1$). However, the occurrence of this trace manifests that both Alice and Bob get their expected evidences *without* the intervene of TTP. As if Alice or Bob tries to contact TTP for recovery, they will just obtain error message instead of evidences. Therefore, this trace does not reveal the involvement of TTP. By the above analysis, we can draw a conclusion that our extended CEM protocol satisfies TTP transparency.

## 6 Related Work

It has been acknowledged that formal verification is important for security protocols, because of the seriousness of security flaws. In this paper, we use the technology of model checking to check automatically whether a given model of CEM protocols satisfy some given specifications. To our knowledge, the literature of formal verifications of CEM protocols includes the works of Kremer et al. [13], Cederquist et al. [8] and Abadi and Blanchet [1].

Kremer et al. [13] propose an approach for modeling and analysis of CEM protocols using model checker Mocha. The advantage of using Mocha is that it allows to model CEM protocols with concurrent game structures, and specify properties in ATL, a temporal logic with game semantics. Therefore, Mocha is well suited for checking properties such as fairness, timeliness and effectiveness that can be naturally interpreted with game semantics. For similar reasons, Mocha has been

used for other fair exchange protocols [10,23]. Besides Mocha, the $\mu$CRL toolset, together with CADP which acts as an back-end, has also used to analyse CEM protocols automatically. Cederquist et al. [8] design an optimistic CEM protocol and check both safety and liveness properties using $\mu$CRL toolset. The desired properties are specified using $\mu$-calculus. There exists another way to check CEM protocols, which is proposed by Abadi and Blanchet [1]. Their protocol is specified using the applied pi calculus. Taking the protocol specification as input language, the verifier ProVerif automatically checks the property secrecy. As to fairness, it is not checked fully automatically, but with some manual proofs.

## 7 Conclusion

We have formally verified the protocol [15], an extension of the key chain based CEM protocol [9] by Cederquist et al. to cover an additional requirement TTP transparency. The verification was taken in two steps. First, we checked fairness, effectiveness and timeliness properties, using the model checker Mocha. Then we have modelled the protocol in a process algebraic language $\mu$CRL and used its toolsets together with CADP to check TTP transparency. Our analysis showed that the protocol achieves the design goals.

The way to formalize TTP transparency in this paper abstracts a lot from the underlying cryptographic techniques and the ability of the adversary. In the future, we would like to investigate a more appropriate approach, for example, it is interesting to see whether we can interpret TTP transparency using static equivalence in the applied pi calculus [2]. Another direction is to extend the protocol furthermore, to cover other design goals such as stateless TTP and accountability.

## References

[1] Abadi, M. and B. Blanchet, *Computer-assisted verification of a protocol for certified email protocol*, Science of Computer Programming **58** (2005), pp. 3–27.

[2] Abadi, M. and C. Fournet, *Mobile values, new names, and secure communication*, in: *Proc. 28th Symposium on Principles of Programming Languages (POPL)* (2001), pp. 104–115.

[3] Alur, R., T. A. Henzinger and O. Kupferman, *Alternating-time temporal logic*, Journal of ACM **49** (2002), pp. 672–713.

[4] Alur, R., T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani and S. Tasiran, *Mocha: Modularity in model checking*, in: *Proc. 10th International Conference on Computer Aided Verification (CAV)*, LNCS **1427** (1998), pp. 521–525.

[5] Asokan, N., M. Waidner and M. Schunter, *Optimistic protocols for fair exchange*, in: *Proc. 4th ACM conference on Computer and Communications Security (CCS)* (1997), pp. 7–17.

[6] Blom, S. C. C., J. R. Calame, B. Lisser, S. M. Orzan, J. Pang, J. C. van de Pol, M. Torabi Dashti and A. J. Wijs, *Distributed analysis with $\mu$CRL: a compendium of case studies*, in: *Proc. 13th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS **4424** (2007), pp. 683–689.

[7] Blom, S. C. C., W. J. Fokkink, J. F. Groote, I. A. van Langevelde, B. Lisser and J. C. van de Pol, *D$\mu$CRL: A tool set for analysing algebraic specifications*, in: *Proc. 13th Conference on Computer Aided Verification (CAV)*, LNCS **2102** (2001), pp. 250–254.

[8] Cederquist, J., R. Corin and M. Torabi Dashti, *On the quest for impartiality: design and analysis of a fair non-repudiation protocol*, in: *Proc. 7th Conference on Information and Communications Security (ICICS)*, LNCS **3783** (2005), pp. 27–39.

[9] Cederquist, J., M. Torabi Dashti and S. Mauw, *A certified email protocol using key chains*, in: *Proc. 3rd Symposium on Security in Networks and Distributed Systems (SSNDS)* (2007), pp. 525–530.

[10] Chadha, R., S. Kremer and A. Scedrov, *Formal analysis of multi-party contract signing*, Journal of Automated Reasoning **36** (2006), pp. 39–83.

[11] Fernandez, J.-C., H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu and M. Sighireanu, *Cadp - a protocol validation and verification toolbox*, in: *Proc. 8th Conference on Computer-Aided Verification (CAV)*, LNCS **1102** (1996), pp. 437–440.

[12] Hwang, R.-J. and C.-H. Lai, *Efficient and secure protocol in fair e-mail delivery*, WSEAS Transactions on Information Science and Applications **5** (2008), pp. 1385–1394.

[13] Kremer, S. and J.-F. Raskin, *A game-based verification of non-repudiation and fair exchange protocols*, Journal of Computer Security **11** (2003), pp. 399–429.

[14] Liu, Z., "Extending a Certified Email Protocol with TTP Transparency and its Formal Verification," Master's thesis, University of Luxembourg (2010).

[15] Liu, Z., J. Pang and C. Zhang, *Extending a key-chain based certified email protocol with transparent TTP*, in: *Proc. 6th IEEE/IFIP Symposium on Trusted Computing and Communications (TrustCom)* (2010), to appear.

[16] Markowitch, O. and S. Kremer, *An optimistic non-repudiation protocol with transparent trusted third party*, in: *Proc. 4th International Conference on Information Security (ICISC)*, LNCS **2200** (2001), pp. 363–378.

[17] Micali, S., *Certified email with invisible post offices* (1997), an invited presentation at the RSA'97 conference.

[18] Micali, S., *Simple and fast optimistic protocols for fair electronic exchange*, in: *Proc. 22th Annual Symposium on Principles of Distributed Computing (PODC)* (2003), pp. 12–19.

[19] Nenadić, A., N. Zhang and S. Barton, *Fair certified e-mail delivery*, in: *Proc. 19th ACM Symposium on Applied Computing (ACM-SAC)* (2004), pp. 391–396.

[20] Onieva, J. A., J. Zhou and J. Lopez, *Multiparty nonrepudiation: A survey*, ACM Computing Surveys **41** (2008), pp. 1–43.

[21] Wang, G., *Generic non-repudiation protocols supporting transparent off-line TTP*, Journal of Computer Security **14** (2006), pp. 441–467.

[22] Zhang, F., R. Safavi-Naini and W. Susilo, *Efficient verifiably encrypted signature and partially blind signature from bilinear pairings*, in: *Proc. 5th International Conference on Cryptology in India (INDOCRYPT)*, LNCS **2904** (2003), pp. 71–84.

[23] Zhang, Y., C. Zhang, J. Pang and S. Mauw, *Game-based verification of multi-party contract signing protocols*, in: *Proc. 6th International Workshop on Formal Aspects in Security and Trust (FAST)*, LNCS **5983** (2009), pp. 186–200.