Front.Comput.Sci.
DOI

RESEARCH ARTICLE

Design and Formal Verification of a CEM Protocol with Transparent TTP

Zhiyuan LIU [1], **Jun PANG** (✉)[2], **Chenyi ZHANG**[3]

1 School of Management Science and Engineering, Shandong Normal University, Jinan 250000, China
2 Faculty of Science, Technology and Communication, University of Luxembourg, Luxembourg 1359, Luxembourg
3 School of Information Technology and Electrical Engineering, University of Queensland, QLD 4072, 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. Cederquist et al. proposed an optimistic certified email protocol, which employs key chains to reduce the storage requirement of the trusted third party (TTP). We extend their protocol to satisfy the property of TTP transparency, using existing verifiably encrypted signature schemes. An implementation with the scheme based on bilinear pairing makes our extension one of the most efficient CEM protocols satisfying strong fairness, timeliness, and TTP transparency. We formally verify the security requirements of the extended protocol. The properties of fairness, timeliness and effectiveness are checked in the model checker Mocha, and TTP transparency is formalised and analysed using the toolsets μCRL and CADP.

Keywords fair exchange, CEM protocols, fairness, TTP transparency, formal verification

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, in order to prove the participation of the other party.

As a special class of fair exchange protocols [1], 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) might be introduced to take charge of the whole procedure and to provide undeniable records of submission (from the sender) and delivery (to 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. A better solution, so called *optimistic* protocols [2], 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. In such situations, a TTP may digitally sign some pieces of information, which will be used later as evidences to guarantee that the protocol ends in a fair state. 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. A typical structure of an optimistic CEM protocol consists of an *exchange* sub-protocol, an *abort* sub-protocol and a *recovery* 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

Received month dd, yyyy; accepted month dd, yyyy

E-mail: jun.pang@uni.lu

contact a TTP to deal with awry situations.

We assume weaker attackers than the Dolev-Yao adversaries [3], in that we do not allow an attacker to block channels, forge messages or impersonate other users. Instead, we assume *resilient* channels between the communicating parties, so that every message is guaranteed to arrive at its receiver eventually, which is especially critical to satisfaction of the fairness requirement. In practice, resilient channels can be guaranteed by physical devices, or be approximated in non-resilient networks by means of software such as the SSL protocol for Internet users. Our work is based on a general concept of resilient channel that has abstracted away all detailed implementations that are mentioned above. Forgeries of messages are handled by the assumption on the crypto strength used by the protocols. We also assume appropriate authentication mechanisms that forbid impersonation of users to happen in our scenario. More specifically, in our attacker model we focus on the dishonest (or malicious) behaviours of users in CEM protocols. A dishonest user may send out a message if he gets enough information for generating the message. He can even send messages after he is supposed to stop. He can send out a wrong message, or withhold a message that he is required to send out at a certain point. Furthermore, a dishonest user may quit at any time, or refuse to stop at a point where his role in the protocol is required to stop. This type of attacker model is generally treated in the analysis of fair exchange protocols, e.g., see [4–12], which allows us to focus on the actual protocol design aiming to the properties of fairness, timeliness and TTP-transparency, and thus significantly reduces the complexities in the modelling phase when using tools Mocha and μCRL/CADP.

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. Besides, in many situations, an institution does not necessarily keep the up-to-date signatures or affidavits from all trusted services (especially when a TTP, who is trusted by the two parties involved in the protocol, may *not* be trusted by an external judge who is to verify the presented evidences). 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 [13], followed by a number of works [14–22], in which different cryptographic schemes are used to achieve TTP transparency, such as interactive proof of knowledge on the encrypted signatures, Schnorr-like signature schemes, and RSA-based encryption schemes.

In this paper, we focus on the development of a CEM protocol with a transparent TTP. Our starting point is the key chain based protocol of Cederquist et al. [23]. The use of key chains is to reduce TTP's storage requirement. Our study exposes a weakness in the original protocol, for which we propose a fix. Later we extend Cederquist et al.'s protocol to satisfy TTP transparency, adopting a recently introduced verifiably encrypted signature scheme [24]. We are able to show, by a detailed comparison, that our protocol is one of the most efficient CEM protocols satisfying *TTP transparency*, in addition to the other important properties such as *strong fairness*, *timeliness*, and *effectiveness*. Furthermore, we show that our protocol satisfies the desired properties, by incorporating formal verification techniques. The finite-state model checker Mocha [25] 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 [26]. The verification of properties expressed in ATL corresponds to the computation of winning strategies. Another toolset μCRL [27, 28] is used for TTP transparency, which requires a comparison of observable traces in various situations. The μCRL toolset has the ability of generating state spaces that can be visualised and manipulated by the toolbox CADP [35] which acts as a backend of μCRL. Preliminary results in this paper have been reported [36, 37].

Contributions. Our contributions are bifold. First we improve the work of Cederquist et al., by fixing a weakness in that version of the protocol and extending it to support TTP transparency. We also measure the complexity of our protocol in terms of timing consumption comparable to RSA signatures. The other contribution is the formal verification of the security properties in Mocha and μCRL/CADP. In particular, to the best of our knowledge it is the first formal analysis of TTP transparency in a symbolic way.

Structure of the paper. We introduce security properties for CEM protocols in Sect. 2. The CEM protocol using key chains is briefly described in Sect. 3. Our extension with transparent TTP and its informal analysis are detailed in Sect. 4. We compare our proposed protocol with some state-of-the-art CEM protocols supporting TTP transparency in Sect. 5. Formal verification of our protocol with Mocha and μCRL is presented in Sect. 6. We conclude the paper in Sect. 7.

2 Security Requirements

A CEM protocol needs to protect a participant who is *honest*, i.e., his behaviour strictly follows the protocol specifications. To this point, for the sake of readability, we write Alice for the sender and Bob for the receiver of an email. We assume the communication channels are *resilient*, in the sense that every message is guaranteed to reach its destination eventually. The following properties are typically required for an optimistic CEM protocol. There are more properties, such as *confidentiality*, *stateless TTP*, *accountability*, and *high performance*, which we do not discuss in this work.

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.

Fairness. Honest Alice (Bob) will get her (his) evidences, provided that the other party gets the evidence from her (him). The evidences can be used to convince an external judge (who is not TTP) 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 *strong fairness* if every judgement on Bob's (Alice's) non-repudiation can be made solely and independently from Alice's (Bob's) evidences, i.e., it does not necessarily involve TTP, nor the participation of Bob (Alice). If besides Alice's (Bob's) evidences, either TTP or Bob (Alice) needs to be contacted during the judgement, the protocol only satisfies *weak fairness*.

TTP transparency. If a protocol runs successfully to the end, then the evidence that each participant obtains is of the same format regardless of whether TTP is involved in the protocol execution or not.

3 A CEM Protocol using Key Chains

We describe the certified email protocol proposed by Cederquist et al. [23]. It makes use of key chains to reduce TTP's storage requirement. Once a key chain is initialised between two communication parties, the initiator can use any key within the chain to encrypt messages. Each exchange that uses the protocol to deliver an email (which may involve a number of message passings) 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. We focus on the main idea of the protocol, with its details available in the original paper [23].

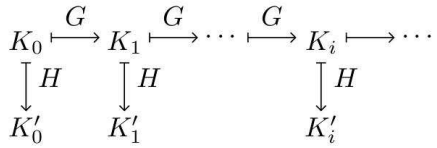

Fig. 1 A key chain.

We use $\{M\}_k$ to denote a message m encrypted with a symmetric key k, and $(M)_P$ to denote party P's signature on message M. In practice a signature is always applied on a hashed value, usually by a user's private (or sometimes called, secret) key.

3.1 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 [17] or a separate key [19] 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 secret keys, involved parties, hash values of email content and so on [38]. The amount of information that TTP needs to store blows up especially when there are a huge number of protocol executions running in parallel, some of which are between the same pair of sender and receiver.

To reduce the TTP's burden of storing too much information, the protocol [23] uses *key chains*. A chain of keys is a sequence of keys $K'_0, \ldots, K'_n$ (see Fig. 1), 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 (κ is a key domain). H and G functions are non-commutative, i.e., given an $H(K_i)$ for which K_i is unknown, it is infeasible to compute $H(G(K_i))$.

3.2 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 $\langle A, B, K_0, \star\rangle$ in her database indicating whether the key chain has been established. If yes, TTP just ignores this request. Otherwise, TTP will choose a session identity sid, send a $\mathsf{cert} := (A, B, sid)_T$ to Alice, and store $\langle A, B, K_0, sid\rangle$ in her database.

3.3 Exchange sub-protocol

The i^{th} protocol round in a protocol session sid is described below. The round number i is initially 0 and then can arbitrarily grow, and Alice increments i after each round.

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

where $\mathsf{EOO}_M := (B, T, i, sid, h(K'_i), \{M\}_{K'_i})_A$, $\mathsf{EOR}_M := (\mathsf{EOO}_M)_B$, $\mathsf{EOR}_{K'} := (A, K'_i, \{M\}_{K'_i})_B$ and h is just an ordinary hash function.

At first, Alice sends out message 1^{ex} to Bob. After receiving this, Bob checks the correctness of the signature on EOO_M and cert. If both are correct, Bob then commits himself to receiving the email by sending out message 2^{ex}. When Alice receives 2^{ex}, she checks the signature on EOR_M. If correct, Alice will send out K'_i to Bob. Upon receiving the key, Bob checks whether this key matches the hash value of the key that he received in message 1^{ex}. If yes, Bob decrypts the email and sends out a confirmation $\mathsf{EOR}_{K'}$ to indicate that he has received the key and the email.

3.4 Recovery sub-protocol

Both Alice and Bob have the right to run recovery sub-protocol by showing EOR_M. The recovery sub-protocol is mainly run with the aim of acquiring key K'_i or evidence $\mathsf{EOR}_{K'}$ with the help of TTP. Typically, Alice runs the recovery sub-protocol when she sends out key K'_i while not receiving message 4^{ex}, and Bob runs it when he sends out EOR_M while not receiving K'_i.

After receiving a recovery request from a party $p \in \{A, B\}$ of the form:

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

where f_r is a flag used to identify the recovery request, TTP checks several things such as correctness of signatures, identities, entries for the key chain. If all checks succeed, TTP can retrieve K_0 and verify whether $h(H(G^i(K_0)))$ matches $h(K'_i)$. If yes, TTP looks up the status of round $status(i)$, to check whether round i has been resolved or aborted. Essentially, if $status(i)$ has not been set, TTP will set it as $h(\{M\}_{K'_i})$ and send back a *recovery token* $(A, B, h(\{M\}_{K'_i}), K'_i, i, sid)_T$ to the requester. If the round is aborted ($status(i) = \mathsf{a}$), TTP will send back an *abort token* $(A, B, h(\{M\}_{K'_i}), \perp, i, sid)_T$. If the status is different from $h(\{M\}_{K'_i})$ or any of the above tests fails, TTP will send back an *error* message in the form of $(\mathsf{error}, (\mathsf{error}, m^r)_T)$, where m^r is the content of the message in step 1^r. This error message indicates a misbehaviour and P can quit the protocol round.

3.5 Abort sub-protocol

Only Alice can abort, if presumably the current protocol round has not yet been recovered. Typically, Alice may abort 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 abrt is Alice's signature on the abort request. After receiving this request, TTP checks several things such as 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 round is 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.

3.6 Evidences and dispute resolution

When a dispute occurs, two parties can provide evidences to an external judge. For each protocol round i, EOO (evidence of origin) desired by Bob consists of

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

EOR (evidence of receipt) desired by Alice consists of

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

if it is obtained by running the exchange sub-protocol. If Alice uses the recovery/abort sub-protocol, then EOR_M and $\mathsf{EOR}_{K'}$ will be replaced by the recovery token. In this case, EOR has the form of

$$A, B, T, M, i, sid, K'_i, \mathsf{cert}, (A, B, h(\{M\}_{K'_i}), K'_i, i, sid)_T.$$

As already remarked [23], the protocol is not TTP transparent, due to the fact that an observer can tell whether TTP was involved by simply checking EOR.

3.7 A vulnerability of the protocol

We found a vulnerability in the protocol. This vulnerability is mainly due to the form of $\mathsf{EOR}_{K'}$ that does not include any information about the current protocol round *i*. An $\mathsf{EOR}_{K'}$ in such form can be reused in different protocol rounds, which causes a breach on fairness.

Fig. 2 depicts a scenario where dishonest Alice breaks strong fairness of honest Bob by reusing $\mathsf{EOR}_{K'}$. This attack requires multiple protocol rounds, which is sketched as follows. Alice first initiates an exchange *i* by sending out 1^{ex}, in which she uses K'_{i+1} instead of K'_i to encrypt the message, and gets the corresponding EOR_M, then does nothing for round *i*. Alice initiates another round $i+1$ with Bob, and behaves honestly in order to acquire correct EOR_M and $\mathsf{EOR}_{K'}$ for round $i+1$. The attack from Alice is based on the fact that $\mathsf{EOR}_{K'}$ used in both rounds *i* and $i+1$ are of exactly the same form, $(A, K'_{i+1}, \{M\}_{K'_{i+1}})_B$. At this moment, Alice has acquired all the necessary evidences for round *i*, leaving Bob in an unfair state. If Bob initiates a recovery sub-protocol, TTP will send back nothing but an error message because of the mismatch between $h(K'_i)$ and $h(K'_{i+1})$. As a result, for round *i*, strong fairness is broken. In order to fix this problem, we decide to revise $\mathsf{EOR}_{K'}$ to be of the form $(A, i, K'_i, \{M\}_{K'_i})_B$, by adding the current protocol round number *i*.

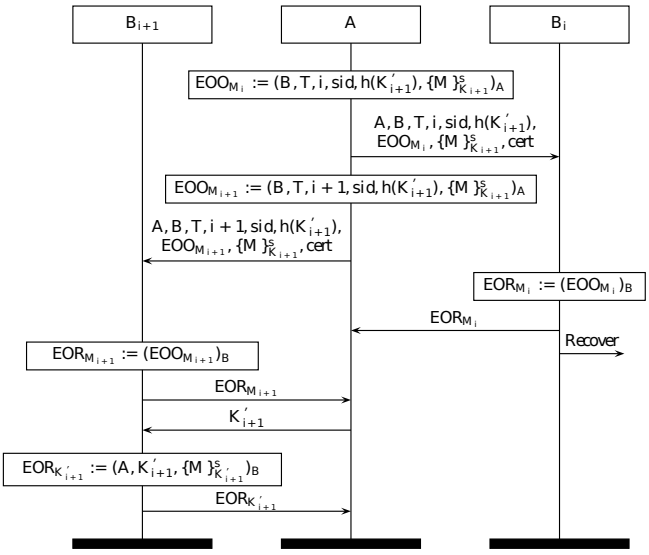

Fig. 2 A vulnerability on the CEM protocol using key chains [23].

4 Protocol Design

We present an extension of the protocol in the previous section to support transparency of TTP. Our approach requires the usage of a *verifiably encrypted signature scheme* to encode Bob's commitment to receive the email in message 2^{ex}.

Notations. 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 $(M)_{B|T}$.

Exchange sub-protocol. The modified exchange sub-protocol is as follows:

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

where $\mathsf{EOR}_M^{\frac{1}{2}} := (\mathsf{EOO}_M)_{B|T}$. After receiving EOO_M, Bob sends out his partial signature on 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 EOR.

Abort and recovery sub-protocols. Alice is allowed to abort provided that she has sent out message 1^{ex}, but has not received message 2^{ex} from Bob. Once honest Alice and Bob contact TTP, they are not allowed to continue the exchange sub-protocol.

Alice is allowed to launch the recovery sub-protocol provided that 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

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

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 EOO_M and EOR_M ($\mathsf{EOR}_M^{\frac{1}{2}}$), (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 look up $status(i)$ for round *i*.

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

$$2^r.\ T \rightarrow B : K_i'$$
$$3^r.\ T \rightarrow A : \mathsf{EOR}_M$$

- 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.

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

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

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

where m^r is the whole message received in step 1_A^r or 1_B^r.

Evidences and dispute resolution. When a disputation occurs, two parties can provide evidences to an external judge. For each protocol round i, EOO (evidence of origin) desired by Bob consists of

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

EOR (evidence of receipt) desired by Alice consists of

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

4.1 Security Analysis

As a special feature, the key chain provides an opportunity for Alice and TTP to have a predefined infinite list of symmetric keys. We assume that K_0, the seed of the chain, is a secret between Alice and TTP during related protocol executions. We restrict our attention to a single protocol round. For multiple rounds a weakness related to key chains has been identified and fixed, but the same vulnerability does not apply in the revised protocol in which $\mathsf{EOR}_{K_i'}$ is no longer used. In the following we informally justify that the revised protocol satisfies the claimed security properties.

Non-repudiation and fairness. If in round i Alice possesses EOR_M, we need to show that Bob must receive $\{M\}_{K_i'}$, K_i' and EOO_M in the same round i. There are three cases.

1. Alice receives EOR_M from Bob in message 4^{ex}, i.e., only the exchange sub-protocol is launched and it successfully runs to the end, during which Bob obtains EOO_M, $\{M\}_{K_i'}$ and K_i'.
2. Alice receives EOR_M from message 3^r by launching the recovery sub-protocol herself, i.e., sending out 1_A^r. Then Alice must possess $\mathsf{EOR}_M^{\frac{1}{2}}$ from Bob's message 2^{ex}, which means Bob must have received 1^{ex} and obtained both EOO_M and $\{M\}_{K_i'}$. Then in the recovery sub-protocol, TTP must have sent message 2^r from which Bob obtains K_i'.
3. Alice receives EOR_M from message 3^r by Bob launching the recovery sub-protocol. Then Bob must have received 1^{ex} and have shown EOO_M to TTP, i.e., Bob obtains both EOO_M and $\{M\}_{K_i'}$, and in message 2^r Bob receives K_i' from TTP.

Furthermore, in case of a dispute, EOR_M alone from Alice is sufficient to prove that Bob has received M.

If in round i Bob possesses M, EOO_M and K_i', we need to show that (1) Alice must receive EOR_M in the same round i, and (2) Alice is the true sender of M. We know Bob can only receive $\{M\}_{K_i'}$ and EOO_M from 1^{ex}. There are two cases.

1. Bob receives K_i' from Alice, then the exchange sub-protocol runs at least up to message 3^{ex}. Bob may send 4^{ex} to Alice which contains EOR_M. If Bob does not send out 4^{ex}, Alice can always get EOR_M from TTP by launching the recovery sub-protocol.
2. Bob receives K_i' from TTP in the recovery sub-protocol. No matter who lauched the recovery sub-protocol, Alice gets EOR_M in message 3^r (from TTP).

As to the authenticity of the message M, Bob is able to convince every third party that M is indeed from Alice by verifying Alice's signature on EOO_M, after extracting M from $\{M\}_{K_i'}$ with K_i. Note that K_i' is also verified as its hashed value is contained in EOO_M too. Since presenting $(M, \mathsf{EOO}_M, K_i')$ is sufficient for Bob to prove that M is originally from Alice, together with the above EOR_M case for Alice, the protocol satisfies *strong fairness*.

Effectiveness. Suppose both Alice and Bob are honest, so that they faithfully follow the protocol in round i, and no error occurs, e.g., there is no significant network delays. It is obvious that only the exchange sub-protocol is launched, and it will stop at a state in which Alice obtains EOR_M, and Bob obtains both M and EOO_M.

Timeliness. In round i Alice can always launch the abort sub-protocol after she sends out message 1^{ex}, so that TTP will send back either an abort token or EOR_M depending on whether a recovery message has already arrived at TTP or not. Bob can launch the resolve sub-protocol any time after he receives message 1^{ex} and will get either an abort token or

K_i', depending on the communication between Alice and TTP. The resilient channels between TTP, Alice and Bob guarantees that the above procedures terminate in a timely manner.

Transparency of TTP. If the exchange sub-protocol successfully runs to the end, Alice's evidence is EOR_M, and Bob's evidences are M, EOO_M and K_i'.

- Suppose the recovery sub-protocol is launched by Alice, then Alice must have message 2^{ex} which contains $\mathsf{EOR}_M^{\frac{1}{2}}$, and Bob must have message 1^{ex} which contains $\{M\}_{K_i'}$ and EOO_M. If TTP successfully verifies EOO_M and $\mathsf{EOR}_M^{\frac{1}{2}}$, TTP will convert $\mathsf{EOR}_M^{\frac{1}{2}}$ into EOR_M and send it back to Alice. Consequently TTP sends K_i' to Bob. In this case both Alice and Bob have the same evidences as only the exchange sub-protocol is launched.
- Suppose the recovery sub-protocol is launched by Bob, then Bob must have message 1^{ex}. If TTP successfully verifies EOO_M and EOR_M, TTP will forward EOR_M to Alice, and send K_i' to Bob, so that they get the same evidences in this case too.

In the next section, we discuss a particular signature scheme and our motivation to implement it in our protocol.

4.2 Verifiably Encrypted Signature Schemes

There is a variety of fair exchange protocols with verifiably encrypted signatures (some of which are also called convertible signatures) existing in the literature. Some of the earliest, such as Asokan et al. [39], Bao et al. [40], Boyed and Foo [41], and Camenisch and Damgård [42], apply interactive proof of knowledge on the encrypted signatures, such that more message exchanges are required in the protocols. Several later approaches use Schnorr-like signature schemes to wrap up signatures, such as by Ateniese and Nita-Rotaru [16], or RSA-based encryption schemes, such as by Markowitch and Kremer [14]. The GPS+RSA scheme used in [14] has been shown an attack by Cathalo et al. [43]. Another attack on fairness proposed by Bao [44] is applicable on several of the signature schemes discussed in [45]. More recently, pairing algorithms for solving the Decision Diffie-Hellman problem in Gap Diffie-Hellman groups have been introduced to generate verifiably encrypted signatures [24, 46]. We briefly sketch the scheme of [24] below.

Let G_1 be a cyclic additive group generated by P with prime order q, and G_2 be a cyclic multiplicative group with the same order q. Let $e : G_1 \times G_1 \rightarrow G_2$ be a pairing operation satisfying *bilinearity*, i.e., $e(aX, bY) = e(X, Y)^{ab}$ for all $X, Y \in G_1$ and $a, b \in Z_q$. The signature scheme is as follows. Suppose G_1, G_2, P, e, q and H are all publicly available, where $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is a cryptographic hash function and λ is the length of the (private) key. A user sets up $x \in Z_p$ as his secret key, and $X = x \cdot P$ for the corresponding public key. A signature on message M is $S = \frac{1}{H(M)+x} \cdot P$. To verify the signature, one only needs to check if $e(H(M) \cdot P + X, S) = e(P, P)$, note that $e(H(M) \cdot P + x \cdot P, \frac{1}{H(M)+x} \cdot P) = e(P, P)^{(H(M)+x) \cdot \frac{1}{H(M)+x}}$, by bilinearity. To produce a verifiably encrypted signature, suppose TTP has private key $y \in Z_q$ and public key $Y = y \cdot P \in G_1$, the new signature on M becomes $S' = \frac{1}{H(M)+x} \cdot Y$, with TTP's public key replacing G_1's group generator P. To verify one only needs to check if $e(H(M) \cdot P + X, S') = e(P, Y)$. TTP is able to get the true signature S by computing $y^{-1} \cdot S'$, where y^{-1} is the inverse of TTP's private key y (in G_2). Note that when applying this scheme to our extended protocol, both $e(P, P)$ and $e(P, Y)$ can be precomputed, thus reduces computation cost for the whole session.

We choose the algorithm in [24] since it requires fewer pairing operations than the algorithm in [46]. Moreover, there exist efficient pairing algorithms that implements pairing operations on elliptic curve-based point groups consuming time comparable to that of the RSA signatures of the same security level. It was studied in [47] that one 256-bit (prime field) pairing operation takes about 15 million clock cycles on a Core 2 Duo processor, which is the most expensive operation in the signature scheme.[1)] According to [48], a 3072-bit RSA encryption (with a small exponent) takes about 620,000 and a decryption takes about 28.6 million cycles on a Core 2 Duo processor.[2)] In practice, 3072-bit RSA signatures are of comparable security strength to 256-bit pairing-based signatures.

5 Comparison

Our protocol supports TTP transparency, i.e., on the completion of a protocol run, the final structure and contents of the evidences possessed by both parties do not reveal whether TTP has intervened in the protocol or not. There are a number of CEM protocols in the literature (e.g. [14–22]) that supports the transparency of TTP, as listed in Tab. 1. The protocol presented in this paper is the only one that satisfies strong fair-

[1)] One 256-bit (prime field) pairing operation takes roughly 43 times as much as that of one point scalar multiplication [47], therefore, we ignore low cost operations such as point scalar multiplication and inverse operation in such signature schemes.

[2)] If we implement the exponentiation algorithm by using Chinese Remainder Theorem, it will take roughly 22 million cycles with the `mpz_powm_sec()` function on a Core 2 Duo processor [48], which is still of comparable speed.

ness, timeliness and TTP transparency with a relatively low cost, to our knowledge.

In the above table, if the correctness of a protocol does not depend on any class of signature schemes, we write down that the protocol is *generic*. In such cases, the particular signature scheme is irrelevant, and the usage of verifiably encrypted signatures or convertible signatures is not required. We use "strong*" to indicate that it is claimed in the paper that the protocol satisfies strong fairness, but there exist attacks in the literature showing that the claim is invalid. All the protocols in the table satisfy TTP transparency, but they differ on other security properties such as timeliness and fairness. We also make comparisons on the number of messages as well as the computational costs as required by the protocols. We write "*#msg*" for the number of messages in the exchange sub-protocol and "*#op*" for the amount of computation equivalent to the number of RSA signature operations, i.e., we interpret other cryptographic operations as the number of RSA signatures referring to the best existing algorithms in the literature.

5.1 Timeliness

Only three protocols support timeliness (those of Wang [19], Markowitch and Kremer [14] and ours). In most cases the lack of timeliness is due to the fact that Alice is not allowed to abort after the first message. This design may trap Alice in a deadlock state, waiting forever on Bob's reply, without any effective ways to escape. Micali's protocol [17] satisfies weak timeliness by using a *cut-off* time, which indicates a deadline moment to resolve in a protocol run, in order to prevent endless waiting. However, this might cause problems if Alice and Bob cannot correctly estimate time differences between their local clocks and TTP's clock. Furthermore, in a real situation such a mechanism might enforce Bob to contact TTP as early as possible instead of replying to Alice if Bob is keen to proceed the current run.

5.2 Fairness

All the protocols except Wang's satisfy (or claim to satisfy) strong fairness. Wang's protocol [19] is not strongly fair, since when Alice is presenting Bob's EOR from the second message, an external judge has to contact either TTP or Bob in order to confirm that Alice has not aborted in the current run. If Alice has successfully aborted before Bob launches the recovery sub-protocol, and Alice has received the second message, Bob will not be able to obtain the key if Alice refuses to send out the third message. Micali's protocol [17] and Imamoto and Sakurai's protocol [15] are vulnerable to replay attack if Bob colludes with an outsider [49]. Both protocols in the work of Markowitch and Kremer [14] are shown to be unfair by Gürgens et. al [38], in the way that if Bob colludes with an outsider, he is able to gain access to the message M without sending EOR_M back to Alice, by recovering the other protocol run with the outsider.[3)] Moreover, the GPS scheme used by the second protocol in [14] has been proved insecure by Cathalo et al. [43]. Nenadić et al.'s protocol applies a particular RSA based verifiable encryption scheme, which has been shown that Bob is able to send an invalid partial signature which is undetectable by Alice and which is not recoverable by TTP [20]. Nevertheless, the protocol presented in [20] is also identified with a similar attack by Hwang and Lai [22]. So far there exist no attacks on the fairness of Ateniese's protocol [16], Hwang et al.'s protocol [22] and Liang et al.'s protocol [21].

5.3 Efficiency

We concentrate on the amount of computation time that is involved in the exchange sub-protocol. The overloads of the abort and recovery sub-protocols are not considered, as such events are supposed to occur rarely. We define *one operation* as one 3072-bit RSA signature operation. As to the RSA scheme, the most time-consuming operation is modular exponentiation, and the ratio of the time taken for a modular exponentiation operation to the time taken for a single modular multiplication is linearly proportional to the exponent's bit length [50]. Therefore, we ignore single modular multiplications and the less time consuming algorithms such as symmetric encryption/decryption and hashing in protocols. For pairing operations, in practice one 256-bit pairing operation can be faster than generating one 3072-bit RSA signature. As a conservative estimation we assume that verifying one pairing-based signature, which is the most time-consuming operation in pairing-based signature schemes, also takes *one operation*. We omit the time used to generate a pairing-based signature as well as that used to verify an RSA-based signature in the analysis. For generic protocols we assume the RSA 3072-bit signature is used. In practice, they may choose faster schemes such as those of 256-bit elliptic curve cryptography (ECC) signatures.

From Tab. 1, we conclude that the first three generic schemes are the most efficient, since they only need three messages in the exchange sub-protocol and at most 4 operations in computation. Nevertheless, none of them achieves

[3)] This attack does not work on our key-chain based approach, because every key chain is uniquely associated to a pair of sender and receiver. Bob and the colluding party are unable to recover from TTP unless Alice is involved.

Protocol	Scheme	Fairness	Timeli.	*#msg*	*#op*
IS02 [15]	generic	strong*	No	3	4
Micali03 [17]	generic	strong*	weak	3	4
Wang06 [19]	generic	weak	Yes	3	2
MK01 [14]	RSA-based	strong*	Yes	4	7
AN02 [16]	RSA-based	strong	No	4	7
NZB04 [18]	RSA-based	strong*	No	4	4
MLCL06 [20]	RSA-based	strong*	No	4	8
HL08 [22]	RSA-based	strong	No	4	6
LCLQ08 [21]	bilinear pair	strong	No	4	5
Our protocol	bilinear pair	strong	Yes	4	3

Table 1 An overview of CEM protocols satisfying TTP transparency.

TTP transparency, strong fairness and timeliness at the same time. As to the other four RSA based protocols, the number of operations varies from 4 to 8. In our protocol, it takes time equivalent to only 3 operations, since only 3 pairings are required (Bob needs to verify Alice's signature in message 1^{ex}, Alice needs to verify Bob's signature in message 4^{ex} and Bob's encrypted signature in message 2^{ex}). A signing operation in pairing-based scheme takes negligible amount of time. From Tab. 1, only our protocol achieves all the three desirable properties – strong fairness, TTP transparency and timeliness – with a relatively low cost.

6 Protocol Verification

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. In this section, we intend to put our analysis one step further, by incorporating formal verification techniques.

It has been acknowledged that formal verification is important for security protocols, because of the seriousness of security flaws. In this section, we apply model checking to automatically verify 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. [4], Cederquist et al. [51] and Abadi and Blanchet [52].

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 [25] allows specification of models as concurrent game structures, and verification of properties in ATL (Alternating-time Temporal Logic) [26] formulas with game semantics, which is suitable for checking properties such as *fairness*, *effectiveness* and *timeliness*. For the analysis of TTP *transparency*, our main idea is to compare execution traces containing evidences acquired in different situations. This methodology needs to put multiple traces together, which is not supported in most of the existing model checkers, including Mocha. Therefore, a process algebraic language μCRL and its toolset [27, 28] are used.

6.1 Mocha and μCRL

Mocha [25] 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, since properties such as fairness and timeliness specify a user's ability to enforce certain outcomes. 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 [25]. Assuming a finite set Π of propositions, an ATL formula is one of the following:

- p for propositions $p \in \Pi$.
- $\neg\phi$ or $\phi_1 \vee \phi_2$, where ϕ, ϕ_1, and ϕ_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 ϕ, ϕ_1 and ϕ_2 are ATL formulas.

ATL formulas are interpreted over the states of a concurrent game structure that has the same propositions and play-

ers [26]. 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 $true\,\mathcal{U}\phi$.

μCRL is a language for specifying distributed systems and protocols in an algebraic style. The μCRL language and its toolset have been applied to the analysis of distributed systems, e.g., see [29–31] and security protocols in particular, e.g., see [32–34]. 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: δ represents deadlock, and τ represents the internal action. $\mathsf{p.q}$ denotes sequential composition, it first executes p and then q. $\mathsf{p+q}$ denotes non-deterministic choice, meaning that it can behave as p or q. Summation $\sum_{\mathsf{d:D}} \mathsf{p(d)}$ provides the possibly infinite choice over a data type D. The conditional construct $\mathsf{p} \lhd \mathsf{b} \rhd \mathsf{q}$, with b a boolean data term, behaves as p if b and as q if not b. Parallel composition $\mathsf{p||q}$ interleaves the actions of p and q; moreover, actions from p and 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 p of actions from the set H into δ. Additionally, the hiding operator $\tau_I(\mathsf{p})$ turns all occurrences in p of actions from the set I into τ. The μCRL tool set [27, 28] is a collection of tools for analysing and manipulating μCRL specifications. The μCRL tool set, together with the CADP tool set [35], which acts as a backend for the μCRL tool set, features visualisation, simulation, LTS generation and minimisation, model checking, theorem proving and state-bit hashing capabilities.

6.2 Verification in Mocha

We first give a sketch of the modelling techniques in Mocha's specification language and discuss how it can be used to describe our extended CEM protocol. Mocha also provides a way to express the security properties of interest in ATL formulas. A more detailed report can be found in [53].

6.2.1 Modelling the protocol in Mocha

Each participant is modelled as a player (in a game), with the description of its behaviours using the guarded command language of Mocha. On the top level we build two models for each participant i, P_i and P_iH, to represent the dishonest and honest behaviours of that participant, respectively. The model P_iH for honest behaviour is strictly in accordance with the protocol, i.e., it strictly follows what the player is supposed to do as specified in the protocol. The dishonest model P_i allows the player to cheat, such as sending out a wrong message, or withholding a message the participant is required to send out at a certain point. Furthermore, a dishonest model may quit at any time, or refuse to stop at a point where its role in the protocol is required to stop. Note that we do not model outside intruders, as the environment in which the CEM protocol runs assumes that potential attacks are only from dishonest participants.

Communication is modelled by using shared variables, as an abstract way of representing message passing. This enables us to focus on the main design mechanism of the designed protocol and limit the models of the protocol to a feasible size for the model checker Mocha. Evidences ($\mathsf{EOO}$and EOR), key and Emails are encoded as boolean variables which are initialised as false and updated by its sender. We model the action of sending out an evidence, or a message by a guarded command in which the sender resets the corresponding variables as true at the time the message is sent out. In the model for honest participant P_iH, the guard consists of all the conditions that needs to be satisfied strictly according to the protocol, and the following action command represents sending of the messages as specified in the protocol. In the dishonest model P_i, more messages (in the form of guarded commands) are allowed at each point of time, however the guard still needs to contain information that is necessary to make its following message passing possible. (That is, all components of a message need to be available before the message is constructed and sent.) As in each transition step, the system nondeterministically picks up one guarded command among all the enabled ones, this way of modelling allows the dishonest model to exhaustively and repeatedly generate all

possible (both 'legal' and 'illegal') behaviours.

List. 1 gives the Mocha code describing the behaviours of honest Alice. (In this language, comment lines are starting with '—') At first, Alice can do idle actions after she initiates a protocol round by sending out 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 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 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 realised by modelling a boolean variable *A_contacted_T*. Finally, Alice can stop if she receives final 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 model the honest behaviours of Bob.

Listing 1 Extracted honest model of Alice

```
-- idle action 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
    [] ~pa_stop & ~A_contacted_T
       & pb_halfeorm & ~k
       -> k':=true
-- (3) Alice stops
    [] ~pb_stop & ~A_contacted_T
       & pb_eorm & ~pa_rece_eorm
       -> pa_rece_erom':=true
-- (4) Alice sends abort request
    [] ~pa_stop & ~A_contacted_T
       & pa_eoo & ~pb_halfeorm
       -> A_contacted_T':=true;
          A_abort_req':=true
-- (5) Alice stops after receiving abort token
    [] ~pa_stop & A_contacted_T
       & T_abort_send_A
       -> T_abort_token_A':=true;
          pa_stop':=true
-- (6) Alice sends recovery request
    [] ~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
    [] ~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 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 EOR_M or an abort token from TTP.

Listing 2 Extracted dishonest model of Alice

```
-- idle actoin while not stopped
    [] ~pa_stop & pa_eoo ->
-- Alice stops
    [] ~pa_stop & pa_eoo
       -> pa_stop':=true
-- (1) Alice sends EOO
  --sends correct hashed key
    [] ~pa_stop & ~pa_eoo
       & ~hk & ~hke
       -> pa_eoo':=true; hk':=true
  --sends incorrect hashed key
    [] ~pa_stop & ~pa_eoo & ~hk & ~hke
       -> pa_eoo':=true; hke':=true
-- (2) Alice sends key
    [] ~pa_stop & ~k -> k':=true
```

```
-- (3) Alice sends abort request
    [] ~pa_stop & pa_eoo
       -> A_abort_req':=true
-- (4) Alice sends 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 recov. 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 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

```
-- (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
```

Note that we also build a two-round protocol model which can be used to represent multiple email delivery communications, and it is based on the one-round protocol model. Details can be found in [53].

6.2.2 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 in which Alice has obtained EOR and Bob has received EOO and M without the involvement of TTP.

$$\mathit{effectiveness} \equiv (\langle\langle P_aH, P_bH\rangle\rangle\Diamond\,(\mathsf{EOO} \wedge M \wedge \mathsf{EOR}))$$

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:

$$\mathit{timelinessP}_a \equiv \forall\Box\,(\langle\langle P_aH\rangle\rangle\Diamond\, P_a\_stop)$$

$$\mathit{timelinessP}_b \equiv \forall\Box\,(\langle\langle P_bH\rangle\rangle\Diamond\, P_b\_stop).$$

where P_aH and P_bH represent the honest Alice and Bob, and P_a_stop (P_b_stop) represents that Alice (Bob) has reached a termination state of the protocol.

Fairness. A protocol is fair for honest Alice (P_aH) if the following is satisfied: whenever Bob obtains P_aH'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:

$$\begin{aligned}\mathit{fairnessP}_aH \equiv \forall\Box\,((\mathsf{EOO} \wedge M)\\ \Rightarrow \langle\langle P_aH\rangle\rangle\Diamond\,(\mathsf{EOR})).\end{aligned}$$

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.

$$\begin{aligned}\mathit{fairnessP}_bH \equiv \forall\Box\,((\mathsf{EOR})\\ \Rightarrow \langle\langle P_bH\rangle\rangle\Diamond\,(\mathsf{EOO} \wedge M)).\end{aligned}$$

6.2.3 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.

6.3 Verification in μCRL

In this section, we give sketchs on how we model the protocol in μCRL, and discuss how to check TTP transparency of the protocol in μCRL. The detailed models and analysis can be found in [53].

6.3.1 Modelling the protocol in μCRL

Each μCRL specification consists of two parts, 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 verifiable encryption of messages.

In our model, we abstract some data types from the protocol, which are Bool, Key, Number, Item, Player, Status and Message. Sort Bool has the same meaning as the normal boolean type. Item is a simple data type with a constructor $d1$, which represents the email content. As our extended CEM protocol is a key chain based protocol, sort Key is modelled to represent the keys that belong to a key chain. For simplicity, we just set two constructors for it. Correspondingly, sort Number is also defined to model the protocol round number. Moreover, to specify the protocol, we assume that there are three processes which are Alice, Bob and TTP respectively. Each of them is assigned with a unique identity (A, B or T), which is described in sort Player. TTP is an important player, which should be impartial. After receiving abort or recovery request, TTP will behave honestly according to his record $\mathit{status}(i)$, for which we define a sort Status.

sort	Status	
func	aborted,recovered,empty	→ Status
map	eq: Status × Status	→ Bool
var	s1:Status	
rew	eq(s1,s1)	=T
	eq(aborted,recovered)	=F
	eq(recovered,aborted)	=F
	eq(empty,aborted)	=F
	eq(empty,recovered)	=F

As the behaviour part of the model is mainly specified by the exchange of messages, defining an appropriate data type for message is necessary, and sort Message we defined is the type that meets our requirements. The constructors for this sort are boolm, itm, player, keym, pair, hash, sign, vesign, enc

and num. Since communications between participants are modelled by messages, the constructors boolm, itm, player, num and keym are defined to change the corresponding data type into sort Message. For example, keym(k1) represents the action of transforming k1 with type Key into Message. Many operations in the protocol are also specified by means of message, such as signing and verifiably signing messages, encryption, etc. The constructor sign has parameters Player and Message, which is used to model signing actions. For example, sign(A,m) means that player A has signed the message m using his private key. Another important action is how to partially sign the message that can be verified by everybody. The verifiably encrypted signature is formed by using signer's private key and TTP's public key. Therefore, the constructor versign is defined with parameters Player and Message. An example vesign(A,T,m1) shows out the verifiably encrypted message signed by player A using player T's public key. Finally, the constructor pair is defined to connect messages.

sort Message
func boolm:Bool → Message
itm:Item → Message
player:Player → Message
num:Number → Message
keym:Key → Message
hash:Message → Message
sign:Player × Message → Message
vesign:Player × Player × Message → Message
enc:Key × Message → Message
pair:Message × Message → Message

There exists two functions eq and keq for sort Message, of which eq is used to compare whether two messages are the same, and the outcome is a boolean type. For example, in order to compare whether the two signed messages are the same, we have the following equation:

eq(sign(p1,m1),sign(p2,m2)) =
if(eq(p1,p2),eq(m1,m2),F).

First, it will judge whether the two messages are signed by the same player, and if so, a further comparison of messages are conducted, or else, it will produce false as an outcome. Another function keq is used to check whether the given key is the right key for a particular protocol round. Normally, it is used by TTP when dealing with recovery request. We omit the detailed definitions of these two functions in the sort specification of Message.

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 [54]. The three models are combinations of (1) honest Alice and honest Bob, (2) honest Alice, malicious Bob and TTP, and (3) malicious Alice and honest Bob and TTP.

Participants are linked 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 μCRL, 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. For example, we define an action for initiator Alice of sending a recovery request to TTP in the form of sendT(A,recover,T), where A and T are the identities of Alice and TTP respectively, recover is of data type Message. Similarly, recvT(T,recover,A) represents the action of receiving a recovery request from Alice. In this way, we can define the behaviours of participants by actions (act) parameterised with data. The main communications are defined as follows. com represents the communication between Alice and Bob, and initCom describes the initialisation communication between them. Similarly, we also use comT to specify the communication between Alice (Bob) and TTP. These synchronisations of actions are enforced by the encapsulation operator ∂_H. In μCRL language, this is captured by a list of equations of the form $s \mid r = c$ under the keyword (comm).

comm send | recv=com
sendT | recvT=comT
initSend | initRecv=initCom

The honest and dishonest behaviours of the participants resemble those in the Mocha models. In the following, we present the μCRL models of honest Alice, dishonest Bob and TTP separately. For instance, the behaviours of the initiator (honest) 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_A(A,y,eoo,i,x,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_A(A,y,eorm,i,x,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.

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

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.[4] We need to extend the above process when taking TTP into account to cover when Alice can contact TTP and receive replies from TTP, which we omit in the above specification.

We use two processes, Bob and Bob1, to model the misbehaviours of Bob. Actually, Bob acts as the main process, and Bob1 with parameters Key, Item and Number works as the sub-process for Bob. At the very beginning, Bob waits for the first message from Alice, using an action initRecv(A,EOO,B) to report the receipt of EOO. After that he performs an action init_Bob to represent his involvement in the protocol. Then he moves to process Bob1, which specifies the misbehaviours.

$$\begin{aligned}\text{Bob}= \sum_{\text{x:Key}} \sum_{\text{y:Item}} \sum_{\text{i:Number}} & \text{initRecv(A,eoo,B).}\\ & \text{init\_B(A,i,eoo,B).}\\ & \text{Bob1(x,y,i)}\end{aligned}$$

Bob1 is a process that acts as a core part of process Bob, and it models Bob's misbehaviours as stated before. From the sketch of process Bob1 in below, we can see that malicious Bob has three choices after receiving the first message from Alice. The first one would be that he honestly sends out his verifiably encrypted message through the action send(B,halfeorm,A). In this case, Bob still can choose between whether to receive key from Alice or rerun process Bob1. If he prefers to receive the key, he will first get his evidence evidence_B(B,y,eoo,i,x,A) and then still face two situations, one is to deliver his final EOR, the other is to return to Bob1. The second choice for malicious Bob is directly sending recovery request to TTP, which is represented with sendT(B,recoveryB,T). After that, Bob may receive abort token (recv_abort_B(x,y,i,A,B)), error message (recv_error_B(x,y,i,A,B)), the desired key recvT(T,k,B), or he just re-executes Bob1. Malicious Bob still can perform Bob1 even if he gets abort token or error message, and is supposed to quit the protocol. We also model that Bob can quit the protocol if he obtains all his expected evidences, such as EOO in the first message and key. Moreover, the last choice for Bob is the deadlock, which means he can quit the protocol at any time he wants.

```
Bob1(x:Key,y:Item,i:Number)=
      send(B,halfeorm,A).
          ( recv(A,k,B).
            evidence_B(B,y,eoo,i,x,A).
                  (Bob1(x,y,i)
                  + send(B,eorm,A))
          + Bob1(x,y,i) )
    + sendT(B,recoveryBob,T).
          ((recv_abort_B(x,y,i,A,B)
            + recv_error_B(x,y,i,A,B)).
              Bob1(x,y,i)
          + Bob1(x,y,i)
          + recvT(T,k,B).
            evidence_B(B,y,eoo,i,x,B))
    + τ.δ
```

Similarly, honest Bob and dishonest Alice can be modelled in μCRL as well.

We present the behaviours of TTP with an identity T with parameters Status by process TTP. Since TTP is a fully trusted participant which cannot misbehave, we model it strictly according to the protocol. TTP can deal with recovery request from both Alice and Bob, and abort request only from Alice.

From the sketch of TTP's behaviour below, we can see that the action recvT(B,recoveryB,T) is used to represent receiving recovery request from Bob. In this case, Bob will first check whether the key used in the protocol is the right key in the key chain. If not, an error message (error_B(x,y,i,eorm,A,B)) will be delivered to Bob. If yes, TTP goes on checking his status for this protocol round i. If the status has already been set to be aborted, the abort token will be sent by actions abort_B(x,y,i,A,B). However, if the status is recovered or just empty, the corresponding key is sent out to Bob, and the status will be kept as recovered. Similarly, TTP receives recovery request from Alice by the action of recvT(A,recoveryA,T).

The process of dealing with Alice's recovery request is similar to that of Bob. The main difference lies in the message that sent to Alice if TTP is sure to help in the recovery process. Actually TTP will first abstracts the final EOR from the verifiably encrypted message and

[4] The detailed specifications of the terms eoo, halfeorm and eorm are left out for the clarity of presentation.

then delivers it, which is simply represented by action sendT(T,eorm,A). TTP can also accept Alice's abort request stated by recvT(A,abortA,T). After that, he checks the TTP's status to make decisions. If the status is recovered, then the final EOR will be sent. Or else, abort token will be sent by actions abort_A(x,y,i,A,B), and after that, the status for protocol round i will be kept as aborted.

```
TTP(s:Status)= ∑y:Item ∑x:Key ∑i:Number
  recvT(B,recoveryB,T).
      (error_B(x,y,i,eorm,A,B)
          ◁ not(keq(x,i) ▷
              (abort_B(x,y,i,A,B)
               ◁ eq(s,aborted) ▷
               sendT(T,k,B).
               TTP(recovered)))
  + recvT(A,recoveryA,T).
      (error_A(x,y,i,halfeorm,A,B)
          ◁ not(keq(x,i) ▷
              (abort_A(x,y,i,A,B)
               ◁ eq(s,aborted) ▷
               sendT(T,eorm,A).
               TTP(recovered)))
  + recvT(A,abortA,T).
      (recover_A(x,y,i,halfeorm,A,B)
      ◁ eq(s,recovered) ▷
      (abort_A(x,y,i,A,B).
      TTP(aborted))
```

After modelling the behaviours of honest and dishonest agents and TTP, we put them in parallel to construct the whole state spaces of models, including (1) honest Alice and honest Bob; (2) honest Alice, dishonest Bob and TTP; (3) dishonest Alice, honest Bob and TTP.

6.3.2 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 (i.e., we keep those actions related to presenting evidences and the starting of a protocol round) and reducing the model (i.e., state space minimisation modulo weak trace equivalence), we obtain a trace from the honest system that is depicted in Fig. 3(a), which shows the situation of getting evidences without TTP. Fig. 3(b) describes traces obtained from the model containing honest Alice, dishonest Bob, and TTP. We can find that Fig. 3(b) has an additional trace. Evidences for both traces are of the same form, but the sequence of getting 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. 3(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. 3(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. Note that in Fig. 3 we have omitted the round numbers in action labels. We in fact also checked models with two protocol rounds. The analysis of TTP transparency is carried out in a similar way. Details can be found in [53].

7 Conclusion

We have proposed a TTP transparent CEM protocol, as an extension of Cederquist et al.'s protocol using key chains. To achieve this, we used a verifiably encrypted signature scheme based on bilinear pairing. Comparing to the existing CEM protocols, ours is among the most efficient ones satisfying strong fairness, timeliness, and TTP transparency. We have formally verified the protocol. The verification was taken in two steps. First, we checked fairness, timeliness and effectiveness properties, using the model checker Mocha. Then we have modelled the protocol in a process algebraic language μCRL and used its toolsets together with CADP to check TTP transparency. Our analysis shows that the protocol achieves the design goals.

In this paper, we have checked the protocol with a limited number of rounds. In general, it is a hard problem to verify the protocol with an arbitrary number of rounds. A possible future direction is to study ways of abstraction [55] or to develop new reduction techniques [56] for game-based model checking, in order to analyse models in Mocha with more protocols rounds. Another direction is to use an inductive approach, e.g. [57], to prove correctness of the protocols in a more general setting.

The way to formalise TTP transparency in this paper relies on an abstraction from the underlying cryptographic techniques and the ability of the adversary. In the future, we

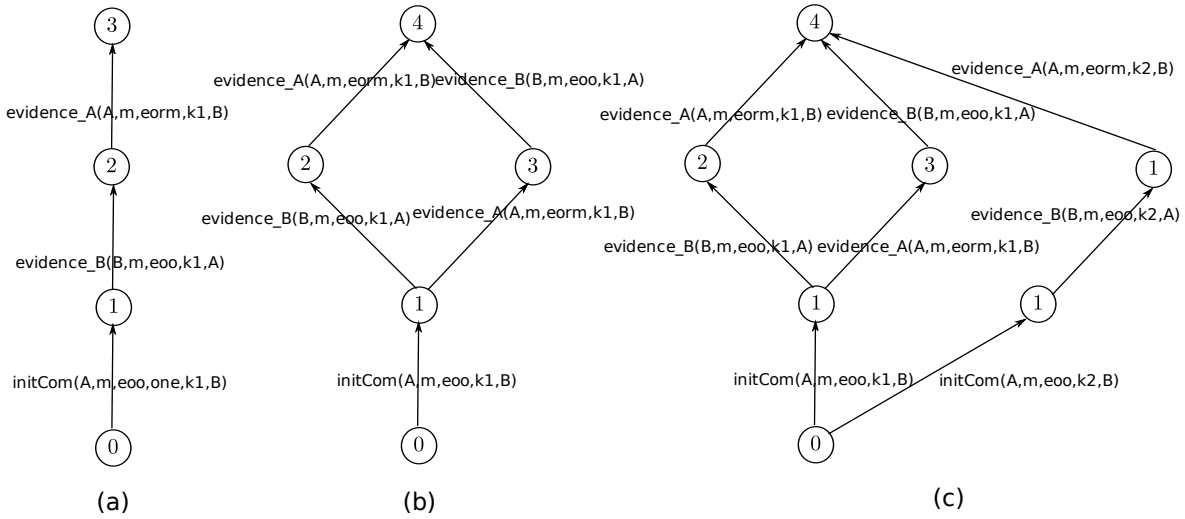

Fig. 3 The obtained traces.

would like to investigate this property in a more sophisticated model, for example, it is interesting to see whether we can interpret TTP transparency using the notion of static equivalence in the applied pi calculus [58]. Another direction is to extend the protocol furthermore, to cover other design goals such as stateless TTP and accountability.

Acknowledgements The authors thank the anonymous referees for their helpful comments. This work was finished while Zhiyuan Liu and Chenyi Zhang were with the Faculty of Science, Technology and Communication at University of Luxembourg. We thank Sjouke Mauw for discussions on CEM protocols and David Galindo for discussions on verifiably encrypted signature schemes. Especially, we thank Johann Großschädl for experimental results on comparing efficiency of pairing operations and RSA signature operations.

References

1. J. A. Onieva, J. Zhou, and J. Lopez. Multiparty nonrepudiation: A survey. *ACM Computing Surveys*, 41(1):1–43, 2008.
2. N. Asokan, M. Waidner, and M. Schunter. Optimistic protocols for fair exchange. In *Proc. 4th ACM Conference on Computer and Communications Security (CCS)*, pages 7–17. ACM, 1997.
3. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
4. S. Kremer and J.-F. Raskin. A game-based verification of nonrepudiation and fair exchange protocols. *Journal of Computer Security*, 11:399–429, 2003.
5. B. Anderson, J. V. Hansen, P. B. Lowry, and S. L. Summers. Standards and verification for fair-exchange and atomicity in e-commerce transactions. *Information Sciences*, 176:1045–1066, 2006.
6. R. Chadha, S. Kremer, and A. Scedrov. Formal analysis of multi-party contract signing. *Journal of Automated Reasoning*, 36(1-2):39–83, 2006.
7. Y. Zhang, C. Zhang, J. Pang, and S. Mauw. Game-based verification of multi-party contract signing protocols. In *Proc. 6th Workshop on Formal Aspects in Security and Trust (FAST)*, volume 5983 of *LNCS*, pages 186–200. Springer, 2009.
8. M. Chen, K. Wu, J. Xu and P. He. A new method for formalizing optimistic fair exchange protocols. In *Proc. 12th Conference on Information and Communications Security (ICICS)*, volume 6476 of *LNCS*, pages. 251-265, Springer, 2010.
9. T. Gaber and N. Zhang. Fair and abuse-free contract signing protocol supporting fair license reselling. In *Proc. 4th IFIP International Conference on New Technologies, Mobility and Security*, 2001.
10. K. Chatterjee and V. Raman. Synthesizing protocols for digital contract signing. In *Proc. 13th Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 7148 of *LNCS*, pages. 152-168, Springer, 2012.
11. D. M. Williams, J. de Ruiter, and W. J. Fokkink. Model checking under fairness in ProB and its application to fair exchange protocols. In *Proc. 9th Colloquium on Theoretical Aspects of Computing (ICTAC)*, volume 7521 of *LNCS*, pages. 168-182, Springer, 2012.
12. Y. Zhang, C. Zhang, J. Pang, and S. Mauw. Game-based verification of contract signing protocols with minimal messages. *Innovations in Systems and Software Engineering*, 8(2): 111-124, 2012.
13. S. Micali. Certified email with invisible post offices, 1997. An invited presentation at the RSA'97 conference.
14. O. Markowitch and S. Kremer. An optimistic non-repudiation protocol with transparent trusted third party. In *Proc. 4th Conference on Information Security (ICISC)*, volume 2200 of *LNCS*, pages 363–378. Springer, 2001.
15. K. Imamoto and K. Sakurai. A certified e-mail system with receiver's selective usage of delivery authority. In *Proc. 3rd Conference on Cryptology in India (INDOCRYPT)*, volume 2551 of *LNCS*, pages 326–338. Springer, 2002.

16. G. Ateniese and C. Nita-Rotaru. Stateless-recipient certified e-mail system based on verifiable encryption. In *Proc. The Cryptographer's Track at RSA Conference 2002 (CT-RSA)*, volume 2271 of *LNCS*, pages 182–199. Springer, 2002.
17. S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proc. 22th Annual Symposium on Principles of Distributed Computing (PODC)*, pages 12–19. ACM, 2003.
18. A. Nenadić, N. Zhang, and S. Barton. Fair certified e-mail delivery. In *Proc. 19th ACM Symposium on Applied Computing (ACM-SAC)*, pages 391–396. ACM, 2004.
19. G. Wang. Generic non-repudiation protocols supporting transparent off-line TTP. *Journal of Computer Security*, 14(5):441–467, 2006.
20. C. Ma, S. Li, K. Chen, and S. Liu. Analysis and improvement of fair certified e-mail delivery protocol. *Computer Standards & Interfaces*, 28(4):467–474, 2006.
21. X. Liang, Z. Cao, R. Lu, and L. Qin. Efficient and secure protocol in fair document exchange. *Computer Standards & Interfaces*, 30(4):167–176, 2008.
22. R.-J. Hwang and C.-H. Lai. Efficient and secure protocol in fair e-mail delivery. *WSEAS Transactions on Information Science and Applications*, 5:1385–1394, 2008.
23. J. Cederquist, 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)*, pages 525–530. IEEE, 2007.
24. F. Zhang, R. Safavi-Naini, and W. Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In *Proc. 5th Conference on Cryptology in India (INDOCRYPT)*, volume 2904 of *LNCS*, pages 71–84. Springer, 2003.
25. R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *Proc. 10th Conference on Computer Aided Verification (CAV)*, volume 1427 of *LNCS*, pages 521–525. Springer, 1998.
26. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of ACM*, 49(5):672–713, 2002.
27. S. C. C. Blom, W. J. Fokkink, J. F. Groote, I. A. van Langevelde, B. Lisser, and J. C. van de Pol. μCRL: A tool set for analysing algebraic specifications. In *Proc. 13th Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 250–254. Springer, 2001.
28. S. C. C. Blom, 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 μCRL: A compendium of case studies. In *Proc. 13th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 683–689. Springer, 2007.
29. J. F. Groote, J. Pang, A. G. Wouters. Analysis of a distributed system for lifting trucks. *Journal of Logic and Algebraic Programming*, 55(1-2): 21-56, 2003.
30. W. J. Fokkink, J. Pang, J. C. van de Pol. Cones and foci: A mechanical framework for protocol verification. *Formal Methods in System Design*, 29(1): 1-31, 2006.
31. J. Pang, W. J. Fokkink, R. Hofman, R. Veldema. Model checking a cache coherence protocol of a Java DSM implementation. *Journal of Logic and Algebraic Programming*, 71(1): 1-43, 2007.
32. J. Pang. Analysis of a security protocol in μCRL. In *Proc. 4th Conference on Formal Engineering Methods (ICFEM)*, volume 2495 of *LNCS*, pages 396–400. Springer, 2002.
33. S. C. C. Blom, J. F. Groote, S. Mauw, A. Serebrenik. Analysing the BKE-security Protocol with μCRL. In *Proc. AMAST Workshop on Real-Time Systems (ARTS)*, volume 139 of *ENTCS*, pages 49–90. Elsever, 2006.
34. T. Chothia, S. M. Orzan, J. Pang, M. Torabi Dashti. A framework for automatically checking anonymity with μCRL. In *Proc. 2nd Symposium on Trustworthy Global Computing (TGC)*, volume 4661 of *LNCS*, pages 301-318. Springer, 2007.
35. J.-C. Fernandez, 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)*, volume 1102 of *LNCS*, pages 437–440. Springer, 1996.
36. Z. Liu, 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)*, pages 630–636. IEEE Computer Society, 2010.
37. Z. Liu, J. Pang, and C. Zhang. Verification of a key-chain based TTP transparent CEM protocol. In *Proc. 3rd Workshop on Harnessing Theories for Tool Support in Software*, ENTCS 274, pp 51-65. Elsevier, 2011.
38. S. Gürgens, C. Rudolph, and H. Vogt. On the security of fair non-repudiation protocols. *International Journal of Information Security*, 4(4):253–262, 2005.
39. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Proc. 6th Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, volume 1403 of *LNCS*, pages 591–606. Springer, 1998.
40. F. Bao, R. H. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line ttp. In *Proc. IEEE Symposium on Security and Privacy (S&P)*, pages 77–85, 1998.
41. C. Boyd and E. Foo. Off-line fair payment protocols using convertible signatures. In *Proc. 4th Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, volume 1514 of *LNCS*, pages 271–285. Springer, 1998.
42. J. Camenisch and I. Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In *Proc. 6th Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, volume 1976 of *LNCS*, pages 331–345. Springer, 2000.
43. J. Cathalo, B. Libert, and J.-J. Quisquater. Cryptanalysis of a verifiably committed signature scheme based on GPS and RSA. In *Proc. 4th Conference on Information Security (ICISC)*, volume 3225 of *LNCS*, pages 52–60. Springer, 2004.
44. F. Bao. Colluding attacks to a payment protocol and two signature exchange schemes. In *Proc. 10th Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, volume 3329 of *LNCS*, pages 137–144. Springer, 2004.
45. G. Ateniese. Efficient verifiable encryption (and fair exchange) of dig-

ital signatures. In *Proc. 6th ACM conference on Computer and Communications Security (CCS)*, pages 138–146. ACM, 199.

46. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proc. 22th Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.
47. P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. In *Proc. 13th Workshop on Selected Areas in Cryptography (SAC)*, volume 5381 of *LNCS*, pages 35–50. Springer, 2009.
48. J. Großschädl. Personal communications, 2010.
49. S. Gürgens and C. Rudolph. Security analysis of (un-) fair nonrepudiation protocols. In *Proc. 1st Internation Conference on Formal Aspects of Security (FASec)*, volume 2629 of *LNCS*, pages 229–232. Springer, 2002.
50. D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notice of the American Mathematical Society*, 46(2):203–212, 1999.
51. J. Cederquist, 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)*, volume 3783 of *LNCS*, pages 27–39. Springer, 2005.
52. M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email protocol. *Science of Computer Programming*, 58:3–27, 2005.
53. Z. Liu. Extending a certified email protocol with TTP transparency and its formal verification. Master's thesis, University of Luxembourg, 2010.
54. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.
55. T. A. Henzinger, R. Majumdar, F. Y. C. Mang, and J. F. Raskin. Abstract interpretation of game properties. In *Proc. 7th Conference on Statics Analysis Symposium (SAS)*. volume 1824 of *LNCS*, pages 220–239. Springer, 2000.
56. E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(4): 527-550, 2003.
57. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2): 85-128, 1998.
58. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL)*, pages 104–115. ACM, 2001.

Zhiyuan Liu received her master degree in computer science from Shandong University in China and University of Luxembourg. She is currently a lecturer in the School of Management Science and Engineering at Shandong Normal University. Her research fields are verification and network security.

Jun Pang received his PhD in computer science from Free University Amsterdam in The Netherlands. He is currently a research scientist in the Computer Science and Communications research unit at University of Luxembourg. His main research fields are formal methods, security and privacy.

Chenyi Zhang received his PhD in computer science from University of New South Wales in Australia. He is currently a research fellow in the School of Information Technology and Electrical Engineering at University of Queensland. His main research fields are formal methods and security.