J. A. Onieva, J. Zhou, and J. Lopez, “Multi-Party Nonrepudiation: A survey”, *ACM Comput. Surveys*, vol. 41, pp. 5, 2008.

Multi-Party Non-Repudiation: A Survey

JOSE A. ONIEVA
Computer Science Department, University of Malaga, Spain
and
JIANYING ZHOU
Institute for Infocomm Research, Singapore
and
JAVIER LOPEZ
Computer Science Department, University of Malaga, Spain

Abstract: Non-repudiation is a security service that plays an important role in many Internet applications. Traditional two-party non-repudiation has been studied intensively in the literature. This survey focuses on the multi-party scenarios and provides a comprehensive overview. It starts with a brief introduction of fundamental issues on non-repudiation, including the types of non-repudiation service and cryptographic evidence, the roles of trusted third party, non-repudiation phases and requirements, and the status of standardization. Then it describes the general multi-party non-repudiation problem, and analyzes the state-of-the-art mechanisms. After that, it presents in more detail the 1-N multi-party non-repudiation solutions for distribution of different messages to multiple recipients. Finally it discusses the advanced solutions for two typical multi-party non-repudiation applications, i.e., multi-party certified email and multi-party contract signing.

Categories and Subject Descriptors: K4.4 [**Computers and Society**]: Electronic Commerce—*Security*; H4.3 [**Information Systems Applications**]: Communications Applications—*Electronic Mail*; C2.2 [**Computer Communication Networks**]: Network Protocols—*Applications*

General Terms: Security

Additional Key Words and Phrases: multi-party applications, multi-party protocols, non-repudiation

1. FUNDAMENTALS OF NON-REPUDIATION

Repudiation is one of the fundamental security issues existing in paper-based and electronic environments. Dispute of transactions is a common issue in the business world. Transacting parties want to seek a fair settlement of disputes, which brings

---

Authors addresses: Jose A. Onieva, Computer Science Department, ETSI Informatica, Campus de Teatinos, 29071 - Malaga, Spain. mailto: onieva@lcc.uma.es. Jianying Zhou, Institute for Infocomm Research, 1 Fusionopolis Way, 21-01 Connexis, South Tower, Singapore 138632. mailto: jyzhou@i2r.a-star.edu.sg. Javier Lopez, Computer Science Department, ETSI Informatica, Campus de Teatinos, 29071 - Malaga, Spain. mailto: jlm@lcc.uma.es.

the need of non-repudiation services in their transactions. The motivation for non-repudiation services is not just the possibility that communicating parties may try to cheat each other. It is also the fact that no system is perfect, and that different and unexpected circumstances can arise in which two parties end up with different views of something that happened. Network failure during the protocol run is a representative example.

We define a *basic transaction* as the transferring of a message M (e.g. electronic goods, electronic cash or electronic contracts) from user A to user B, and represent this event with the following message flow: $A \rightarrow B : M$. Thus, typical disputes that may arise in a basic transaction with a deadline T could be

- A claims that it has sent M to B while B denies having received it;
- B claims that it received M from A while A denies sending it;
- A claims that it sent M before T while B denies receiving it before T.

Fair non-repudiation can be considered as an extended *fair exchange* problem in which non-repudiability is made an integral requirement of the exchange (in general it may not be required). We can find various instances of the general exchange problem in different types of commercial activities: purchase, contract signing, certified mail or, more generally, in any barter conducted by means of digital networks.

An exchange is said to be *fair* if at the end of the exchange, either each player receives the item it expects or neither player receives any additional information about the others' item. For instance, in payment protocols, fair exchange can ensure that a customer receives the digital goods from a vendor if and only if the vendor receives the payment from the customer.

The features of the transaction will decide the type of non-repudiation services to be deployed. For any non-repudiation services, evidence is a crucial object, and the processing of evidence usually involves the assistance from *Trusted Third Parties* (TTP). There are different activities at each phase of processing. The non-repudiation policy defines the behavior of these activities. Finally, the eventual success of non-repudiation depends upon technical and legal supports.

Non-repudiation is, thus, one of the essential security services in computer networks defined by the ITU in X.813 [ITU-T X.813 1996]. Following, we establish the characteristics of this security service and survey the progress of standardization of non-repudiation in general. Further in this survey, we analyze this service when multiple entities are involved.

1.1 Specific Non-repudiation Services

Non-repudiation services help the transacting parties to settle possible disputes over whether a particular event or action has taken place in a transaction. We define a *non-repudiation protocol* as a message flow in which entities exchange digital evidence in order to provide such non-repudiation services.

In an electronic transaction, message transfer is the building block and there are two possible ways of transferring a message (see figure 1).

- The originator O sends the message to the recipient R directly; or

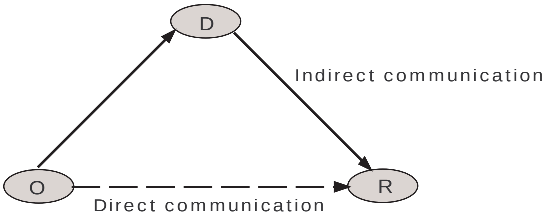

Fig. 1. Models of message transfer

- The originator O submits the message to a *delivery agent* D which then delivers the message to the recipient R.

In the direct communication model, because the originator and the recipient eventually do not trust each other, the originator is not sure that the recipient will acknowledge a message it has received. On the other hand, the recipient will only acknowledge messages it has received. In order to facilitate a fair exchange of a message and its receipt in which neither party will gain an advantage during the transaction, a TTP will usually be involved. Of course, the extent of the TTP's involvement varies among different protocols, which allows to provide a protocol distinction.

To establish the accountability for the actions of the originator and the recipient, the following non-repudiation services are required.

- **Non Repudiation of Origin (NRO)** is intended to protect against the originator's false denial of having originated the message. *Evidence of Origin (EOO)* is generated by the originator or a TTP on its behalf, and will be held by the recipient.
- **Non Repudiation of Receipt (NRR)** is intended to protect against the recipient's false denial of having received the message. *Evidence of Receipt (EOR)* is generated by the recipient or a TTP on its behalf, and will be held by the originator.

In the indirect communication model, a *delivery agent* is involved to transfer a message from the originator to the recipient. In order to support the settlement of possible disputes between the originator and the delivery agent or between the originator and the recipient, the following non-repudiation services are required.

- **Non Repudiation of Submission (NRS)** is intended to provide evidence that the originator submitted the message for delivery. *Evidence of Submission (EOS)* is generated by the delivery agent, and will be held by the originator.
- **Non Repudiation of Delivery (NRD)** is intended to provide evidence that the message has been delivered to the recipient. *Evidence of Delivery (EOD)* is generated by the delivery agent, and will be held by the originator. Similarly, we should be aware that evidence provided by this service cannot be used to make further deductions about the delivery status without some sort of assumption on the communication channel.

1.2 Evidence

The evidence is the data or information that can be used if a dispute arises. It can be either generated and stored by the local user or by a third party. Its format depends on the cryptographic mechanisms agreed in the service, such as *digital signatures* (public key cryptography) and *secure envelopes* (secret key cryptography). Whichever the format is, this evidence has to be composed on common information that helps to clearly identify a transaction and thus resolve a possible dispute in a more deterministic way. Some of these common elements are:

- Non-repudiation service to which evidence is related
- Non-repudiation policy identifier
- Originator identity
- Recipient identity
- Third party identity if evidence generator differs from the originator
- Message or a digital fingerprint
- Message identifier
- Information needed for verifying evidence (i.e. digital certificate, symmetric secret key info) if it is not publicly available
- TTP's identifier and role (see section 1.3) when involved in the service
- Unique evidence identifier
- Time information (time and date in which evidence was generated, expiry date, . . .). If this data is certified by a *Time Stamping Authority* (TSA) it could include a time-stamp service identifier.

When a secure envelope is used to provide evidence, data is stamped with a secret key known only by the TTP, thus being the generator and verifier of evidence as requested by the users.

TTP participation can be relaxed through the use of smartcards or manipulation-resistent modules [ITU-T X.813 1996] in which secret keys are properly installed. In this case, the smartcard plays the role of a distributed TTP. The generator smartcard is used for evidence generation while the verifier's one only for validation. The latter one cannot be used to generate evidence with the secret key (even if it is the same one), such that only the user who owns the generator smartcard could have created the evidence. This is achieved by correctly installing the secret key and the module which controls whether the user can use its smartcard for generation or verification. This module is tamper-proof and different for the generator and the verifier such that it performs just one of the two possible functions.

The secure envelope maintains integrity of the information using a digital fingerprint (i.e. hash function) and confidentiality (e.g. symmetric cipher with the secret key).

When a digital signature is used to provide evidence, information is enclosed in a data structure digitally signed by a *Certification Authority* (CA) such that only the CA can sign the data and other participants (recipients and TTP) can verify it. Unforgeable digital signatures provide a clear statement of the essential components of handwritten signatures; namely, a user's ability to sign by itself, a universally agreed verification procedure and the assertion that it is unfeasible (or at least very hard) to selectively forge signatures in a manner that passes the verification process without being detected.

In order to bring all of this into reality, digital signatures used as evidence in a non-repudiation service need an infrastructure backing it up. There will be a third party certifying participants' link between their identity and public key. Only in this way any recipient can verify the digital signature. Digital signatures introduce a new disrupting element in the non-repudiation service, as the link certified by the CA (often referred as *digital certificate*) may have an expiry date. This fact has to be checked when evidence is verified either by the recipient or a TTP (e.g. an adjudicator). If this link has expired, evidence will be valid only if it was generated before. For this reason, time information has to be included in the evidence generated.

In general, it is more efficient, in terms of computation, for users to use secure envelopes with symmetric encryption techniques, since the TTP (or the smartcard) is in charge of the generation/verification process. Nevertheless, in this case,

- Principals have to unconditionally trust a third party for evidence generation and verification;
- TTP's on-line availability is needed in order to participate in the service when requested;
- If users are to relax the TTP participation as stated previously, then they need to use dedicated hardware to avoid the TTP becoming a bottleneck.

So, users would likely prefer to use digital signatures because

- It only needs an implicit trust over the CA computing the digital certificates. But this trust can be relaxed with legal agreements between users and authorities, audited registration processes and a quite advanced standardization [ITU-T X.509 2000];
- Trust imposed over the CA is less critical than in the former case, since it certifies the existence of a binding between a user and a public key, verifying at the same time that this uniquely corresponds to a private key. But this TTP does not need to know the key itself. So, there is no danger of this entity accessing the content or even being able to generate it (as with secure envelopes).

Additionally, Maurer [Maurer 2004] proposed a novel view of digital evidence called *digital declarations*, based on a digital recording of a willful act indicating agreement to a document or contract. This proposal tries to address some of the problems mentioned above that digital signatures bring with them. It also includes new elements in the digital evidence (as willful acts) to augment the concept of evidence, bringing it nearer to the one used in human judgements. Among all the concepts introduced by Maurer, the semantics of certificates is very important. He proposes that when registering the public key, the user must explicitly commit to be liable for signatures with respect to that public key. Evidence confirming this commitment, designated as change has several important implications for us:

- The certificate has absolutely no value as evidence in court, only the commitment declaration does.
- Only the recipient of a signature (evidence) must trust the CA.

- An expiration date stated on the commitment declaration must be interpreted differently. It specifies until when evidence can be presented as valid, regardless of when it was generated. In other words, evidence expires, not public keys. As a consequence of this view, the validity period of evidence should be kept short.
- A commitment declaration cannot be revoked. Revocation of a public key is impossible (not needed).

Actually, with these definitions, the signature seems to be more insecure than in the traditional view when revocation is possible while the commitment declaration is valid. But, on the other hand, it seems to be closer to the business model if we consider the discussed users' liability in the traditional approach [1]. Maurer proposes the concept of delegation signatures (digital signatures assisted by TTPs) to strengthen its security.

Furthermore, this digital declaration and commitments are a new approach to digital evidence with no implications on how non-repudiation protocols handle the evidence.

1.3 Roles of the TTP

One of the main features which allows us to classify the TTPs is its role on a non-repudiation service. A TTP which does not participate actively in the non-repudiation service, i.e., it will be invoked only when there is something wrong in a transaction, is referred to as *off-line TTP*. An *on-line TTP* participates in the generation and verification of evidence throughout the protocol instance. An *in-line TTP* acts as an intermediary in all the interactions among the users. The difference between third parties that are used only in case of exceptions and third parties that are actively involved in a protocol was first explained in [DeMillo and Merritt 1983]. Obviously, the first type is preferred if efficiency is the major concern, but in some situations and e-commerce applications, to have a delivery agent or intermediary could be the best practical solution.

Other roles have appeared as a consequence of research achieved in exchange protocols. These new approaches aim at eliminating the involvement of the TTP completely but need strong requirements; either all involved parties must have the same computational power as in *gradual exchange* or fairness depends on the number of protocol rounds [Markowitch and Roggeman 1999] as in *probabilistic protocols*.

Finally, an additionally existing trusted third party is the *Adjudicator*. This is the party which drives a resolution process to a conclusion depending on evidence presented by the entities and optionally contacting the TTP which participated in the protocol. In order to facilitate its task, a well defined dispute resolution process in accordance with the non-repudiation policy must exist. This dispute resolution process has to take into consideration the legal framework in which it is defined. New or established *On-line Dispute Resolution* (ODR) processes can be used [Brannigan 2004].

---

[1] In the current digital signature laws, the main "hot potato" does not come from the technical aspects but from the users' liability when it does not understand the technical process or this is done without its knowledge.

1.4 Non-repudiation Phases

Non-repudiation services establish accountability of an entity related to a particular event or action to support dispute resolution. Provision of these services can be divided into different phases such as: generation, transfer, verification, storage and dispute resolution.

1.4.1 *Evidence Generation.* Evidence generation is the first phase in the provision of a non-repudiation service. Depending on the non-repudiation service being provided and the non-repudiation protocol being used, evidence could be generated by the originator, the recipient, and/or the TTP. The elements of non-repudiation evidence and the algorithms used for evidence generation are determined by the non-repudiation policy in effect. When NRO and NRR services are required, evidence of origin and receipt are usually generated by the originator and the recipient, respectively, if digital signature is used for evidence generation. When NRS and NRD services are required, evidence of submission and delivery will be generated by a TTP, like a notary or a delivery authority. If a secure envelope is used for evidence generation, it should always be generated by a TTP on behalf of the originator or recipient.

A TTP may also generate and provide supporting evidence in a non-repudiation service. For example, in a fair non-repudiation protocol [Zhou and Gollmann 1996], the notary will digitally sign the message key provided by the originator and make the confirmed message key available to both the originator and the recipient. The confirmed message key will serve as part of non-repudiation evidence to prove that the message key was sent from the originator (via the notary), and is available to the recipient.

1.4.2 *Evidence Transfer.* Evidence transfer is the most challenging phase in the provision of a non-repudiation service. It mainly consists of the sending and reception of evidence among participants. Actually, it represents the core of a non-repudiation protocol. It is greatly influenced by the communication channel properties. The different options are as follows:

- The communication channel is *unreliable*. In this case, data can be lost.
- The communication channel is *resilient* (also called asynchronous network). In this case, data is delivered after a finite but unknown amount of time.
- The communication channel is *operational* (also called synchronous network). In this case, data is delivered after a known, constant amount of time.

An unreliable channel will in most cases be transformed into a resilient channel by the use of the appropriate transport protocol (e.g. retransmissions).

1.4.3 *Evidence Verification.* Newly received evidence should be verified to gain confidence that the supplied evidence will indeed be adequate in the event of a dispute arising. The verification procedure is closely related to the mechanism of evidence generation.

If evidence is generated through a secure envelope, it should be verified by a TTP at the request of the user because the secret key for evidence generation and verification is only held by the TTP. Obviously, the extra communication between the user and the TTP will cause a substantial delay which might be unacceptable for many on-line electronic transactions.

1.4.4 *Evidence Storage.*

Because the loss of evidence could result in the loss of future possible dispute resolution, the verified evidence needs to be stored safely. The duration of storage will be defined in the non-repudiation policy. For extremely important evidence aimed at long term non-repudiation, it could be deposited with a TTP.

1.4.5 *Dispute Resolution.*

Dispute resolution is the last phase in a non-repudiation service. This phase will not be activated unless disputes related to a transaction arise. When a dispute arises, an adjudicator will be invoked to settle the dispute according to the non-repudiation evidence provided by the disputing parties and the non-repudiation policy in effect. This policy should be agreed in advance by the parties involved in the service.

The adjudicator needs to verify the evidence, probably with the assistance from other TTPs, e.g. from a notary when evidence was generated through a secure envelope. Nowadays, different on-line arbitrator platforms [2] exist which allows for dispute resolution being processed through document and evidence transactions as well as the cooperation of on-line parties [3]. The dispute resolution process can either be registered in one of these platforms and use its services or use its own rules for the definition of an on-line arbitrator.

1.5 Non-repudiation Requirements

Different targets of each non-repudiation service may influence the protocol design. Nevertheless, there are several common requirements in the design of a good non-repudiation protocol.

— *Fairness.* Repudiation can only be prevented when each party is in possession of proper evidence and no party is in an advantageous position during a transaction. The reliability of communication channels affects evidence transfer. Moreover, a dishonest party may abort a transaction, which could leave another party without evidence. Various fair non-repudiation protocols with different features have been proposed. Some of them can be found in [Kremer et al. 2002; Gürgens et al. 2003]. Asokan defined two levels of fairness [Asokan 1998]. A protocol fulfills *strong fairness* if when the exchange is completed, A at least can prove to an arbitrator that B has received (or can still receive) the item, without any further intervention from A. On the other hand, a protocol fulfills *weak fairness* if when the exchange is completed for A, it can prove to an arbitrator that B has received (or can still receive) the item, or otherwise an affidavit can be presented to demonstrate that B misbehaved or a network failure occurred.

— *Efficiency* is another criteria. TTPs will usually be involved in non-repudiation services and its involvement will be essential in order to determine the efficiency of the protocol. Fair non-repudiation protocols proposed in [Asokan et al. 2000;

---

[2] Note that these platforms themselves may need to implement a non-repudiation service.

[3] See http://www.ietf.org/html.charters/ltans-charter.html or http://www.disputemanager.com/mediation/what.asp or http://www.dr.bbb.org

Asokan et al. 1997; Zhou and Gollmann 1997; Pfitzmann et al. 1998; Asokan et al. 1998; Markowitch and Saeednia 2001; Micali 2003] meet the criteria of efficiency and are often called *optimistic* protocols. Some authors define this property as *effectiveness*; that is, if no error occurs and no party misbehaves, then the TTP should not intervene.

— *Timeliness* is also desirable in evidence transfer. For various reasons, a transaction may be delayed or terminated. Hence, the transacting parties may not know the final status of a transaction on time, and would like to unilaterally bring a transaction to completion in a finite amount of time without losing fairness.

— *Policy.* This has to perfectly define all the parameters needed by the service, some of which can be: rules for evidence generation and verification, rules for evidence storage, evidence use and the dispute resolution process.

There are optional requirements, and their fulfillment depends on the application itself. If the application requires them, they turn out to be as critical as the common ones previously defined.

— *Verifiability of TTP*: This property adds one level of security to the protocol itself when it does not exist a strong trust relationship among participants with the TTP which collaborates in the protocol. If the TTP misbehaves resulting in a loss of fairness for any participating entity, all harmed parties will be able to prove it to an arbitrator or verifier. It can be very useful during the initial setup of a non-repudiation infrastructure as well as in those scenarios in which the TTP has to be selected by the entities on the fly (e.g. in an ad-hoc network). It usually assumes that when the TTP misbehaves, the rest of the entities are honest.

— *Transparency of TTP*: It also appears in the literature as *invisible TTP*. If the TTP is contacted to help in the protocol, the resulting evidence will be similar to the one obtained in case the TTP is not involved. This is especially important in practical cases, in which an institution does not wish to change the existing processes to accommodate the new signatures or affidavits generated by TTPs. At the same time, this property helps in the privacy of users with respect to the use or not of a TTP during the protocol run.

Unfortunately, these last two properties are more often incompatible (achieving one of them increases the difficulty to fulfill the other one) and a trade-off has to be assumed when designing the protocols.

1.6 Analysis of Standards

ISO and ITU standards provide a guideline for engineering and should reflect the state-of-the-art of science and technology. Non-repudiation is one of the security services in the ISO/OSI security framework, and is especially important for securing electronic commerce. Many efforts have been devoted to the standardization of non-repudiation services and mechanisms. However, some issues have not yet been well addressed.

There are two international standards dealing with non-repudiation: ISO/IEC 10181-4 [ISO/IEC 1996] [4] and ISO/IEC 13888 [ISO/IEC 2004; 1998; 1997]. The

---

[4] Revised by 10181-4:1997 Information technology – Open Systems Interconnection – Security

first one refines and extends the concept of non-repudiation services as described in ISO 7498-2 and provides a framework for the development and provision of these services. In this framework, the goal of non-repudiation and types of non-repudiation services are defined. The basic mechanisms for non-repudiation services and general management requirements for these services are identified. The roles that a *TTP* plays in non-repudiation services are listed. The relationship of non-repudiation services to other security services is explained. As a general framework, this standard does not include specific non-repudiation mechanisms. This remains as an open issue treated in [ISO/IEC 1998; 1997].

ISO/IEC 13888 "Information technology - Security techniques - Non-repudiation" is composed of three parts. The first one [ISO/IEC 2004] [5] serves as a general model for subsequent parts specifying non-repudiation mechanisms using cryptographic techniques. It establishes two main types of evidence, the nature of which depends on cryptographic techniques employed: the *secure envelopes* and *digital signatures* generated by an evidence generator (which can be the user itself) or an evidence generating authority using asymmetric cryptographic techniques.

In [ISO/IEC 1998; 1997], a set of non-repudiation mechanisms based on symmetric and asymmetric cryptographic techniques are identified. All of them are final international standards in the different phases that ISO/IEC apply to its documents. The history of this multipart standard which is being developed by ISO/IEC JTC1/SC27 dates back to August 1991 [ISO/IEC 1991]. Zhou's book [Zhou 2001] analyzes the ISO/IEC 13888 non-repudiation mechanisms, and points out their weaknesses and limitations. It also discusses the problems on defining the roles of time stamps in the ISO/IEC 13888 non-repudiation evidence.

In 2006, and in response to a request of ISO's Subcommittee 27 (SC27) Secretariat, the Working Group 2 (WG2) of the same Subcommittee agreed to revise ISO/IEC 13888-2 as well as ISO/IEC 13888-3. The same happened later with ISO/IEC 13888-1. Though drafts of the three parts have been circulated within WG2, no final documents are available at the moment. But it seems that changes will not be dramatic.

On the other side, ITU defines a general framework for the provision of non-repudiation services in X.813 [ITU-T X.813 1996] similar to ISO/IEC 10181-4. It defines non-repudiation as "the ability to prevent entities from denying later that they performed an action". The non-repudiation framework extends the concepts of non-repudiation security services as described in X.800 and provides a framework for the development of these services.

2. DESCRIPTION OF THE GENERAL MPNR PROBLEM

As commerce applications like e-voting, e-bidding, etc. usually involve several parties, we have focused this survey in the multi-party scenario. For identifying the *Multi-Party Non-Repudiation* (MPNR) problem, we studied several existing approaches of multi-party scenarios in related topics such as fair exchange, contract commitment, etc.

---

frameworks for open systems: Non-repudiation framework.

[5] This document revises ISO/IEC 13888-1:1997, which is withdrawn.

2.1 Definitions

Extracted from the different multi-party applications and protocols, let us define our view of a MPNR scenario.

Definition 2.1. In a general MPNR scenario, $n \mid n > 2$ entities agree to use a non-repudiation protocol for exchanging messages (general or specific purpose) and collecting evidence of the transactions performed for the exchange of those messages.

Of course, different topologies are possible (e.g., one-to-many, ring, mesh), but some of them seem to be more natural than others. For instance sending the same message to several entities is more related to existing Internet applications than one entity receiving the same message from different originators. Nevertheless, all the topologies need to be considered as long as applications exist for them. We should not forget that there are applications in the collaborative and e-learning areas in which many to many messages are a reality. For example, in [Asokan et al. 1996] a *simultaneous payment for receipt* is presented as an instantiation of a many-to-one application, and in [Asokan et al. 1998] a many-to-many contract signing protocol is depicted.

Let us imagine the following scenario which sketches a virtual application for managing market shares. Several, n, users (or machines) which are market share holders meet up in order to bid for each other stocks (some users could share one or several stocks). We can represent the bids as a set of messages $Set_i = m_1, \ldots, m_m$. For a user U_i, there are $n - 1$ entities to which offers from the set can be sent. The same offer can be submitted to all of them or to a subset of the entities. Also different offers can be sent to all or any combination of them. Once the offers are sent, user U_i will wait for a response. This response could be made individually or could require several recipients gathering for replying an offer. This is a typical example of a many-to-many application which can be represented by a binary matrix, in which "1" in the (i, j) position indicates that user U_i sends a bidding message to user U_j.

Each user U_i may need evidence of receipt of the message sent while receivers may need evidence of origin of the offer received. Many other multi-party applications and protocols can be represented using a matrix. For instance, in the case of the *Multi-Party Contract Signing* (MPCS) protocol mentioned above, a matrix in which all elements are "1" except the diagonal represents mathematically the topology used.

This scenario could be seen as a typical multi-party fair exchange scenario as described in [Asokan et al. 1996]. Even though general MPNR and multi-party fair exchange protocols have a common design goal (*fairness*), several differences can be found:

- In a fair exchange protocol each entity offers a priori known item (i.e., something is known about the item a priori but not its precise content) and receives another item, also known a priori. In a multi-party fair exchange protocol one can imagine sending an item to one entity and receiving an item from a different one. In non-repudiation it does not make sense that one entity receives some data and a different entity sends the corresponding receipt of that data.

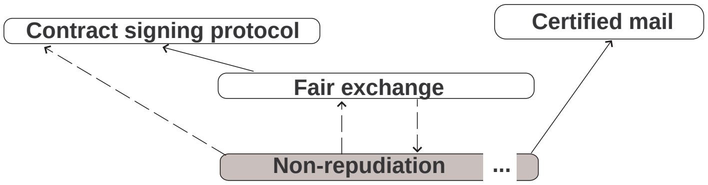

Fig. 2. Non-repudiation service

- With a general message M, non-repudiation is more related to a certified e-mail service in which the receipt has to be sent by the receiver in order to be able to disclose the message received and obtain the evidence of origin for it. Therefore, a ring topology is not applicable in a *general* MPNR service.
- Due to the fact that in a non-repudiation service it is not an exchange of items, MPNR protocols have to continue and finish even if some parties do not reply. Only parties following the protocol correctly should be able to disclose the message and obtain evidence of origin for it.
- In some MPNR applications, the message content is previously revealed and known by the recipients. In these applications it is more important the origin and occurrence of the transmission, that is, the fair exchange of EOO and EOR.
- There is no *exclusion-free* property as defined in [González-Deleito and Markowitch 2002] in MPNR protocols. Since there is no need for a setup phase to agree on participants and items, no danger of excluding participants exists.

The reason of these differences is also due to the classification of fair exchange and non-repudiation as services or applications. Although it seems clear that non-repudiation is a service [6], sometimes it is referred in the literature as an application and the authors compare it with fair exchange. Actually fair exchange can also be considered as a service. While non-repudiation might provide a service to applications like certified electronic mail, fair exchange can provide service to other upper-layer applications as well like payment protocols or digital contract signing. Although non-repudiation should be a security service in this kind of applications, they can be designed without it (see figure 2). Note that this view is not contrary, but complementary, to the one provided in [Markowitch et al. 2002].

Other type of applications in which multi-party protocols appear (either for fair exchange, non-repudiation, contract signing, certified electronic mail or any other evidence-generating exchange) are those in which the participants play different roles in the same application. Imagine an electronic shopping application which involves a customer, a merchant, a credit card company and a delivery company. Different existing two-party non-repudiation protocols could be selected for providing a non-repudiation service to this scenario. Nevertheless, several questions, mainly regarding efficiency, arise. The solution achieved is not optimal for a real application since the correlation among different parties in a unique transaction is lost.

---

[6] In fact, as mentioned in previous section, it has been standardized as a service.

The ground for focusing on the multi-party problem as a different one comes from the fact that the solutions to be provided are different. The first step towards multi-party environments consists of revising the requirements defined in 1.5.

Definition 2.2. Fairness. A multi-party protocol is said to be fair if at the end of the protocol **all honest** parties receive what they expect or none of them receive any valuable information.

It is very important to detail what is *valuable information*. Most existing definitions do not specify the valuable elements and it is not always straightforward to identify these elements. In a general fair MPNR protocol, the recipients either already know the message to be received and thus only evidence are considered valuable elements for the exchange (i.e. they can refuse to run the protocol even if they get the message and we say the protocol fulfills *light fairness*), or they must send a NRR in order to get the message. Thus, in the latter case, instantiating the definition above, we say that a MPNR protocol is *fair* if, at the end of the protocol, either the originator receives NRR and the recipient(s) receive(s) the message and the corresponding NRO or none of them obtains any of these items.

Note that all the participants must be in the same state at the end of the protocol. That is, corrupted parties should receive their outputs if and only if the honest parties also receive them [Lindell 2003].

Definition 2.3. Confidentiality. A multi-party protocol is said to be confidential if only the aimed honest recipients can disclose the message.

This means that the TTP cannot disclose the message either. Nevertheless, though it is a must in multi-party protocols that others do not disclose the message, it is optional whether the TTP is able to do it.

Definition 2.4. Efficiency. A multi-party protocol is said to be efficient if, assuming participating entities of the protocol are honest, the TTP does not intervene.

Different instances of a non-repudiation protocol in which different entities participate could make the TTP become a bottleneck. An optimistic protocol in which an off-line TTP participates only in the case of an exception seems to be the solution, as we already explained for two-party protocols. Nevertheless, many entities could be participating in each instance of a multi-party protocol. In this multi-party environment, even only in case of exception, the TTP needs to act in a light way such that it does not become a bottleneck.

Definition 2.5. Timeliness. A multi-party protocol is said to respect timeliness if all honest entities are able to terminate the protocol in a finite amount of time without losing fairness.

The honest transacting parties may not know the final status of a transaction in time, and would like to unilaterally bring a transaction to completion at any time without losing fairness. It should be noted that from the definition of timeliness, two versions of this property appear:

- *Asynchronous timeliness*: A multi-party protocol is said to respect asynchronous timeliness if all honest entities are able to terminate the protocol at any time without losing fairness. In this case, there are no deadlines for participants in the protocol, but a serious practical implication makes it hard to achieve: for this property to be fulfilled, an infinite state (or at least until evidence expiry date if it is the case) has to be maintained by the TTP. Otherwise, if responsible for distributing evidence, the TTP needs to retry till recipients of evidence acknowledge reception. An alternative solution can be to extend the channel reliability between TTP and users (in this direction).

- *Synchronous timeliness*: A multi-party protocol is said to respect synchronous timeliness if all honest entities are able to terminate the protocol in a finite *and known* amount of time without losing fairness. In this case, deadlines are used and the TTP clock is assumed as the reference time (i.e., users' clocks need to be synchronized with the TTP's clock). Though more difficult for users, the TTP does not need to maintain evidences for long-time periods (*stateless TTP*).

Definition 2.6. Policy. A MPNR protocol needs a complete non-repudiation policy that supports the participation of several entities in possible dispute resolution processes.

As with two entities, this has to define perfectly all the parameters needed by the service. There is no important distinction in multi-party environment except for the fact that multiple disputes and agreements can arise among the participants. The arbiter to judge the dispute needs to be explicitly mentioned in the policy definition.

2.2 State of the Art and Analysis

The first work that appears in general MPNR problems can be found in [Kremer and Markowitch 2000a; Markowitch and Kremer 2000]. Those papers propose generalizations of two protocols: one based on the on-line approach [Zhou and Gollmann 1996] and the other based on an optimistic approach (also known as with off-line TTP) [Kremer and Markowitch 2000b]. As stated before, other related works also existed and are still under research in the field of multi-party fair exchange [Asokan et al. 1996; Franklin and Tsudik 1998; Bao et al. 1999; González-Deleito and Markowitch 2001; Khill et al. 2001; González-Deleito and Markowitch 2002], contract signing [Asokan et al. 1998; Baum-Waidner and Waidner 1998; Garay and MacKenzie 1999; Baum-Waidner and Waidner 2000; Ferrer-Gomila et al. 2001; Baum-Waidner 2001; Chadha et al. 2004; Ferrer-Gomila et al. 2004] and certified e-mail [Ateniese et al. 2001; Ferrer-Gomila et al. 2002]. Due to its practicality, works on the MPNR and *Certified Electronic Mail* (CEM) use a one-to-many topology, whereas multi-party fair exchange and contract signing appears either on mesh or ring configurations.

2.2.1 *Multi-Party Fair Exchange(MPFE).* The general optimistic multi-party fair exchange protocol in [Asokan et al. 1996] is the first work which achieves a scenario with multiple entities in which non-repudiation evidence needs to be generated together with the fair exchange of the items. Asokan et al. proposed the use of a matrix of descriptions as information of the items to be exchanged for all entities. Depending on the type of items (confidential data, public data and payment) and number of entities (one-to-many, etc.), the generic service described can be instantiated as a particular application; e.g., one-to-many topology in which confidential data is exchanged by public data corresponds to a reliable certified broadcast application.

In this way, the basic idea of the generic multi-party fair exchange protocol is that each party signs the expected global description or local view (i.e., the whole description matrix) of the exchange and commits to the items he will send to every other party. If all parties signed the same description matrix, the parties send the promised items. If someone does not receive what was expected, two-party recovery procedures are started with the TTP.

Besides, the authors introduced the notion of *revocable* items (e.g., the payment could be cancelled by the bank even when the order was already transmitted) and *generatable* items (e.g., the bank could order on behalf of one of its clients, a payment to another client). In this protocol, the TTP can guarantee strong fairness if the items involved are either all revocable or all generatable, otherwise weak fairness is achieved.

The resolution process of this protocol, though complete, is not efficient. Having the TTP observing every exchange between two parties when a problem exists and if not sufficient, running recovery resolution processes, is not a practical solution for an application aimed at the Internet.

In [Franklin and Tsudik 1998], the authors also develop a classification of types of multi-party fair exchange schemes and present new protocols which assume the presence of a *semitrusted neutral party* in ring topologies. A malicious semitrusted neutral party must be unable to cheat as long as the other parties remain honest. For achieving fairness the protocol makes use of a special mathematical function f which allows to make checks without revealing content. This is similar to the *verifiable encryption scheme* appeared in some approaches [Bao et al. 1999]. The TTP acts in an in-line manner (i.e., it is involved in every exchange) and does not learn about the items exchanged as well as the topology (it knows it is a ring but does not know the sequence).

In [Bao et al. 1999], the authors propose a multi-party fair-exchange protocol with a ring topology making use of an off-line TTP. In this protocol, no honest party is left in an unfair situation no matter how maliciously the dishonest parties behave. Nevertheless, some parties could be excluded from the ring in the exchange (not affecting however fairness). The main design, which uses a verifiable encryption scheme as a tool for commitment, is linear with respect to the number of rounds ($2n$ rounds) and the TTP needs to contact the originator of the protocol. This makes efficiency an important issue to be improved. In fact, two main improvements are proposed for enhancing efficiency; reducing the number of sequential rounds and avoiding the contact of the TTP with the initiator of the protocol. In [González-Deleito and Markowitch 2001], the protocol is improved such that participants only need to trust the TTP (and not the initiator). Moreover, under certain circumstances, if there are participants excluded from the exchange, they can prove to an external adjudicator that a problem occurred.

Markowitch tackles the exclusion-freeness property in multi-party fair exchange protocols one year later in [González-Deleito and Markowitch 2002]. This time, they demonstrate the exclusion problem in Franklin and Tsudik's protocol. They provide a formal definition of the exclusion-freeness property and propose a multi-party fair exchange protocol with on-line TTP which respects the strongest definition of the property defined.

In [Khill et al. 2001], the authors propose a protocol for multi-party fair exchange with a ring topology. Although it is not optimal, it presents an extensive complexity study with respect to [Asokan et al. 1996]. The protocol consists of three phases in which a matrix of elements are exchanged from the initiator to the next party in the ring (the sequence is predefined). Upon receiving the message, each party checks it and pulls out the information components destined to it from the matrix. It may load the next sequenced components into the matrix with regard to the accepted components at this point. The TTP participates in the exchange in abnormal cases. It tries to recover from the abnormality by observing and controlling all the transferred messages. In those cases each message from a party to the next party is not transmitted directly but relayed by the TTP.

2.2.2 *Multi-Party Contract Signing.*

A particular application of multi-party fair-exchange protocols is MPCS, although MPCS protocols can also be used to design multi-party fair exchange protocols when the signed contract is used as evidence -*verifiable commitment*- which preserves fairness in the exchange. In [Asokan et al. 1998], the authors propose the first optimistic MPCS for synchronous networks. Assuming that the TTP is not corrupted, in the all-honest case only two rounds of communication are needed. In the first round, each party who wishes to sign the contract broadcasts a signed "promise to sign". In the second round, each party who receives all promises from the previous round signs the contract and broadcasts its real signature. Obviously, this works if all parties wish to sign. If at least one party does not wish to sign, it will not send the signed promise, and thus no party will sign in round 2.

If some party cheats, two more rounds are added to the protocol, such that everybody who has all signed promises from round 1 can get them converted into a valid contract by the TTP. If the TTP issues an affidavit, it broadcasts it to all parties in round 4. Thus, each party who did not receive all promises in round 1 waits until round 4. If it receives an affidavit from the TTP, the decision is `signed`, otherwise `failed`. On asynchronous networks, the previous "otherwise" would not be effective, since a party could not decide whether an affidavit was not sent, or just not delivered yet. In fact, due to this synchronicity, termination of the protocol is ensured by the network itself in a fixed number of rounds (four in the all-honest case and six in the worst case). Using this protocol as a building block, Asokan et al. designed a generic multi-party fair exchange and a multi-party certified electronic mail protocol (see next subsection for further reference).

In [Baum-Waidner and Waidner 1998], the first optimistic MPCS for asynchronous networks is presented. Again, assuming that the TTP is not corrupted, $n+4$ rounds of communication and $O(n^2)$ messages are needed in the worst case, taking into account that the number of dishonest parties is not known a priori. It consists of $n + 1$ rounds in the all-honest case. In round 1 each party that starts with `sign` signs a "promise" to sign the contract and broadcasts this promise. In each subsequent round each party collects all signatures from the previous round, countersigns this set of n signatures, and broadcasts them. The result of the (n+1)-nd round becomes the real contract. Three additional rounds are needed if the TTP is contacted.

A party who becomes tired of waiting for some signatures in some round (i.e., there is a local timeout) can call the third party. The TTP analyzes the situation and decides either failed or signed: if the first request received comes from a party in the first round, then the TTP must decide failed (the TTP cannot know whether some parties might have started the protocol with `reject`). If the TTP receives a request from a party in the last round, then the TTP must decide signed because other parties might already have the signed contract. Besides, if the TTP receives multiple requests somewhere in the middle between the first and the last round, the TTP might have to change the decision from failed to signed.

The problem is that the TTP can do this only if all parties that received failed before are probably dishonest. Therefore, the TTP needs $n + 1$ rounds in order to check this behavior. If dishonest parties call the TTP one after the other starting with round 1 then first request must be answered with failed, as already explained. After that, for $i > 2$, the request by party i [7] shows that party $i-2$ to first party are all dishonest: party i calling in round i means that it has seen all messages of round $i - 1$, because otherwise, party i would have called the TTP already then. Those include messages from party $i - 2$ to the first one, which could not exist if those would have stopped in round $i - 2$ to the first, as supposed. Thus, whenever the TTP receives a request for a certain round i ($i \geq 2$) such that it has not answered a request for round $i - 1$ yet, then it can safely switch from failed to signed (since parties who previously obtained failed token are clearly dishonest).

As a consequence of the number of rounds, asynchronous timeliness is achieved; i.e., no deadlines are used and each party can finish the protocol when desired. Using this protocol as a building block, the authors designed a scheme which provides fairness to any general secure multi-party function evaluation protocol. These protocols allow several users to calculate the output of a function f without revealing to the rest of participants their inputs. With the added fairness, dishonest parties have no advantage over the honest parties in learning the function outcome (basically composing this outcome in such a manner that a signed contract protocol is used for exchanging one of the components needed for learning the other component which is the real function output).

In [Garay and MacKenzie 1999], the authors construct a general multi-party optimistic asynchronous contract-singing protocol which requires $O(n^3)$ messages in $O(n^2)$ rounds. The protocol is also *abuse-free*, meaning that at no point can a participant prove to others that he is capable of choosing whether to validate or invalidate the contract. This is the first abuse-free optimistic contract-signing protocol that was developed for $n > 2$ parties. They also showed a linear lower bound of n rounds of any n-party optimistic contract signing protocol.

For this purpose, a crypto tool denominated *Private Contract Signature* (PCS) is used. Upon receiving a PCS, a party convinces himself of its validity, but cannot convince anybody else, and he also knows that the TTP appointed by the signatory can convert it into a regular (self-authenticating) signature. The full formal definition described in [Garay and MacKenzie 1999] presents *invisibility*; i.e., no one can

---

[7] i establishes the order of rounds and a party i is the party requesting the TTP in round i.

determine if a conversion was performed by the original signer or the TTP. This, of course, allows the TTP to be transparent.

Nevertheless, the more important contribution of this work is not the protocol itself, which is rather complicated and not optimal, but the theorem presented at the end of the paper:

> *Any complete and optimistic asynchronous contract-signing protocol with n participants requires at least n rounds in an optimistic run.*

Describing the theorem, Garay *et al.* stated that "... for each party P_i out of n, when it sends a message that can be used (together with other information) by other entities to obtain a valid contract, as the protocol is fair, it must have received in a previous round, a message from the rest of participants in order to be able to get a valid contract too, no matter how others behave (probably with TTP's help)". By an inductive argument, they showed the number of rounds is at least n.

We found another argument which, perhaps, better explains the theorem stated above from a more practical point of view. Bearing in mind the technical requirements for fulfillment of fairness in an asynchronous contract signing protocol, we base our argument on the number of rounds a TTP needs to determine whether a party is misbehaving when requesting resolution (i.e., it requests cancel to the TTP but continues the main protocol):

THEOREM 2.7. *The TTP cannot determine whether a party is misbehaving until round* $= round_{current} + 1$*, since the TTP needs to wait to the next round to see whether this entity cheated and continued the protocol. That means, if* $n - 1$ *dishonest parties exist in the worst case, and each of them requests TTP's cancel sub-protocol in a different round,* n *rounds are the minimum required to satisfy fairness in an asynchronous optimistic contract signing protocol.*

This is the argument used by other solutions to reduce the number of rounds and steps when the number of dishonest parties t, is limited and known a priori. In [Baum-Waidner 2001], by assuming t, the protocols presented achieve an improvement in comparison with the state of the art. The number of rounds is reduced from $O(t)$ to $O(1)$ for all $n \geq 2t + 1$, and for $n < 2t + 1$, it grows slowly compared with the number of rounds in $O(t)$: If $t \approx \frac{k}{k+1}n$ then the number of rounds $\approx 2k$. Nevertheless, in the worst case ($t = n - 1$), the number of rounds is (obviously) still $t + 2 = n + 1$. From a theoretical point of view, the main unrealistic assumption of this work is assuming a known a priori set of dishonest parties. Anyhow, from a practical point of view, these could be protocols with a determined tolerance which could work in real applications in which parties assume the risk on the limited number of dishonest parties (specially in relative trust environments). Furthermore, these protocols could be used in combination with others, such that they are used when at least 1/2 of the participating entities have previously demonstrated their good behavior, and thus, taking advantage of its better efficiency. As these protocols use a threshold, we will refer to them in the rest of this paper as *threshold protocols.*

2.2.3 *Composition of Secure Multi-Party Protocols.* When research in multi-party protocols is conducted, there is another important issue to be taken into account: composition of secure multi-party protocols. In [Lindell 2003], it is demonstrated that obtaining security under (even rather weak notions of) composition can be strictly harder than the stand-alone execution of the protocols. That is, interactions between different executions of the same multi-party protocol more often reduce security. MPNR protocols themselves need to be studied to see how different executions interact with each other and how it affects security. For instance, MPNR protocols might make use of multi-party primitives as broadcasts, and it has been demonstrated that it is impossible to achieve broadcast if a third or more of the parties are corrupted. And this is under parallel self-composition, which is weaker than the concurrent self-composition we foresee for MPNR protocols.

A protocol is said to be *self-composed* if it remains secure when it alone is executed many times in a network. It is *parallel self-composed* if all executions begin at the same time and proceed at the same rate (i.e., in a synchronous fashion). It is *concurrent self-composed* if several executions of the same protocol start and finish in an arbitrary way or determined by the adversary (i.e., in an asynchronous fashion).

Nevertheless, composition of secure multi-party protocols is a large area of research out of the scope of this paper. Furthermore, we can base the design of MPNR protocols in the fact that *Universal Composition* (UC) protocols can be achieved if a reference string is used in the composition (i.e. a setup phase). Thus, we assume this setup phase for allowing the composition of these multi-party protocols, even though we do not raise this issue any more. As an example, in this setup phase, transaction identifiers (or labels) have to be perfectly defined in order to distinguish among different executions of the same protocol and participants must be aware of not being engaged in an exchange in which it has been already participating. Details for protocols for UC realizing any multi-party functionality in the *common reference string model* can be found in Chapter 4 in [Lindell 2003].

Remark 2.8. Including the common reference model and, hence, a setup phase for secure composition of MPNR protocols, produce the exclusion problem. Nevertheless, in MPNR protocols there is no exchange of a priori known items or promises. That is, the recipient does not know from what it has been excluded and therefore, the exclusion-freeness decreases its importance as in fair exchange or contract signing protocols.

2.3 Description of the 1-N MPNR Problem

Several applications, like email and multicast of content in general, present a 1-N topology. In some of these applications, the originator sends a message which should not be revealed unless the recipients confirmed their reception. Let us define what is our view of a 1-N MPNR scenario:

Definition 2.9. A 1-N MPNR application is an instantiation of the general MPNR problem; one entity O is willing to send a (several) message(s) to a set of recipients R such that a subset $S \subseteq R$ obtain the message and evidence of origin if and only if they sent evidence of receipt.

As previously mentioned, Kremer and Markowitch proposed generalizations of two protocols: one based on the on-line approach and the other based on an optimistic approach (also known as with off-line TTP). For both of them, the originator sends an initial commitment (message encrypted with key k) and once a group of recipients reply to this commitment it sends the message to which it was committed to (i.e. it sends the key k which allows the recipients to decipher the expected message) and waits for evidence of receipt. As the same message is distributed, only one key needs to be distributed to those recipients which replied, and a group encryption scheme based on the Chinese Reminder Theorem is used for this purpose [Chiou and Chen 1989].

The on-line based approach uses the TTP in a light-weight manner to distribute the key for each execution of the protocol. In this way, the originator submits the key together with evidence of submission and the TTP publishes this key along with its final signature needed by all entities to complete their evidence. As explained before, only those who behaved correctly can decipher the key and, thus, the message.

In the optimistic approach, and if no exception arises, the TTP does not take part. Otherwise, entities can recover the protocol if needed after a deadline fixed by the originator with respect to the TTP's clock. The originator will contact the TTP when not receiving final evidence from recipients to which it sent the key. Recipients will contact the TTP when not receiving the key while having sent evidence of receipt of the commitment. Once the TTP is contacted for resolving the protocol, and assuming it happens after the fixed deadline, the TTP needs to access an originator database in which recipients who replied are marked. The originator needs to be notified of TTP's successful access and stop immediately the main protocol. Of course, the authors assume that the channel between originator and TTP is bidirectionally resilient. If TTP receives all necessary information, it sends evidence of confirmation to all entities, which substitute evidence not received (maybe due to a channel failure) by participating entities. They showed that the improvement with respect to n parallel executions of a two-party non-repudiation protocol is in terms of number of messages as well as number of needed digital signatures.

For multi-party certified e-mail applications, different protocols can also be found. Asokan et al. instantiated (and optimized some steps) their generic multi-party fair exchange explained in the previous section to design a reliable broadcast application. Besides, Asokan et. al presented an optimistic multi-party certified e-mail protocol as an application of an MPCS protocol presented in [Asokan et al. 1998]. That is, the MPCS protocol is used as a building block. The block can be asynchronous or synchronous depending on the type of network the certified email application is running on. It follows the following schema:

(1) The sender prepares the message M and *tid* (a transaction id) to be sent to all the recipients encrypting it with the TTP's public key. This $cipher = E_{TTP}(M, tid)$ turns out to be a contract to be signed by all parties.

(2) All participants sign the contract using the MPCS building block and *tid* as the contract signing transaction identifier. Only if the final decision of the signing process is *signed* the protocol continues. The signed contract will be the receipt for the sender.

(3) The sender sends to all the recipients the cleartext mail M. If any of the recipients does not receive this message, it contacts the TTP sending the signed

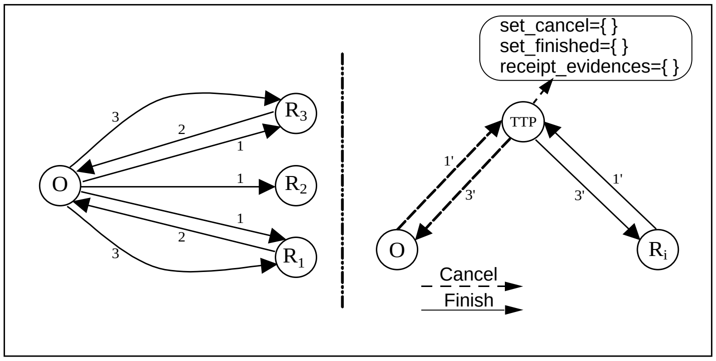

Fig. 3. Ferrer et. al solution for multi-party certified email

contract $(tid, cipher)$. The TTP checks the consistency of tid after decrypting $cipher$. If the contract is valid and decryption succeeds then the TTP signs the cleartext M for the recipient, otherwise it signs a null message.

In the original paper, it can be found the proof based on demonstrating two properties: no information is leaked on M unless the sender has a receipt. And the sender cannot get a receipt without revealing M. In this work, Asokan et al. also define other possible types of multi-party certified e-mail applications with many-to-one and mesh topologies. All of them can be solved following the same schema described above.

The only apparent difference with respect to the general MPNR protocols described in the previous section is that, as the authors indicated, honest recipients should receive the message even if some dishonest parties do not follow the protocol correctly and thus the MPCS block outputs *failed*. The main privacy problem of this solution is that since it uses a MPCS as a building block, each recipient can identify the rest of addressees in the protocol.

With this in mind, Ferrer et al. presented a realistic protocol for multi-party certified electronic mail in [Ferrer-Gomila et al. 2002] which can be seen as an instantiation of a MPNR service. If we compare this protocol with Kremer's optimistic proposition already discussed, the former is more efficient (only three steps in the main exchange which will be the only one executed in the majority of executions) and solves some of the problems found in the latter one.

In this protocol (see figure 3), the sender commits itself to the message, and sends the necessary information (encrypted with the TTP's public key) to open this commitment together with evidence of origin. Then, honest recipients send evidence of receipt and finally the sender opens the commitment and sends the required evidence to those recipients who replied. With respect to the MPNR properties, the protocol is efficient, can be timely terminated thanks to the existence of two sub-protocols (*cancel* and *finish*) and defines a sufficient dispute resolution process. Besides, the TTP behavior is verifiable (unless colluding with other entity, the TTP cannot misbehave without being discovered).

If the sender does not get some receipts from some participants, it initiates a cancel sub-protocol in which those recipients are included in a cancelled set if they have not finished the protocol yet. On the other hand, if some recipients do not receive step 3 of the main exchange protocol, they contact the TTP in order to finish it, and those who have not been cancelled yet by the sender, receive the key to open the commitment and therefore, obtain the cleartext message. Otherwise, they receive a cancel token. In this finish sub-protocol, recipients need to send their receipts to the TTP, which will store and forward them to the sender if required. A weakness can be found in the dispute resolution process: the arbiter needs to contact both disputing parties in order to be able to maintain fairness, i.e., an entity and its evidence could not be sufficient to prove its claim.

We introduce tables I and II in order to summarize the different advantages, properties, disadvantages and functionalities of the fair protocols presented so far, as well as our own solutions.

3. MPNR PROTOCOL WITH DIFFERENT MESSAGES

In the existing MPNR solutions a message can be multicasted to several recipients. In that way no personal and confidential messages can be sent to these parties without loss of privacy. In this section we review the new solutions [8].

3.1 On-line MPNR Protocol for Distribution of Several Messages

Sending (same or different) messages to several recipients could mean a single transaction in a specific application. Therefore, it would be better to store the same key and evidence in the TTP record for every protocol run. In those types of applications, the storage and computation requirements of the TTP are reduced and it will be easy to distinguish between different transactions, regardless of how many entities are involved. An example application which can get advantage of these kind of security protocols is the notification systems. These systems notify different users with customized messages and need evidence of having notified them (e.g., a paper acceptance notification system for scientific conferences). In this type of applications we will distinguish between whether the recipients already know or not the content of the message to be received (see subsection 3.3 for further details).

Some useful notation in the protocol description is as follows:

- O : an originator
- R : set of intended recipients
- R' : subset of R that replied to O with the evidence of receipt
- M_i : message being sent from O to a recipient $R_i \in R$
- n_i : random value generated by O
- $v_i = E_{u_{R_i}}(n_i)$: encryption of n_i with R_i's public key
- k : key being selected by O
- $k_i = k \ xor \ n_i$: a key for R_i
- $c_i = E_{k_i}(M_i)$: encrypted message for R_i with key k_i

---

[8] The work in this section has been partially published in [Onieva et al. 2003] and [Onieva et al. 2004].

Protocols	**App.**	**Properties**	**Advantages**	**Drawbacks**	**Topology**
Asokan'96	MPFE	optimistic, generic service, revocable and generatable items	asynchronous timeliness, NR evidence generated, strong and weak fairness	not efficient	N-N
FraTsu'98	MPFE	in-line TTP	semitrusted neutral party, confidentiality	no exclusion-free	ring
Bao'99	MPFE	off-line TTP, verifiable encryption scheme	asynchronous timeliness	no exclusion-free, not efficient	ring
GDM'02	MPFE	on-line TTP	exclusion-free		ring
Asokan'98	MPCS	optimistic, synchronous	4 rounds	synchronous timeliness	ring
BW'98	MPCS	optimistic, asynchronous	asynchronous timeliness	n+4 rounds, $O(n^2)$ messages	ring
Garay'99	MPCS	optimistic, asynchronous, PCS crypto tool	timeliness, transparent TTP, abuse-free	$O(n^2)$ rounds, $O(n^3)$ messages	ring
BW'01	MPCS	optimistic, asynchronous, threshold	optimal efficiency		ring
KM'00	MPNR	group encryption scheme	light TTP, one key	same message, on-line TTP	1-N
MK'00	MPNR	group encryption scheme	off-line TTP, one key	synchronous timeliness, same message	1-N
Asokan'98	CEM	based on MPCS protocol, a/synchronous	optimistic	no privacy	1-N
Ferrer'02	CEM		optimistic, efficient, timeliness, verifiable TTP	arbitrator needs to contact disputing parties	1-N

Table I. Multi-party protocols' properties summary

- $l_i = h(O, R_i, TTP, h(c_i), h(k))$: label [9] of message M_i
- L' : labels of all the messages sent to R'
- t : a timeout chosen by O, before which the TTP has to publish some information
- $E_{R'}(k)$: a group encryption scheme that encrypts k for the group R'
- $EOO_i = S_O(feoo, R_i, TTP, l_i, t, v_i, u_{R_i}, c_i)$: evidence of origin for R_i
- $EOR_i = S_{R_i}(feor, O, TTP, l_i, t, v_i, u_{R_i}, c_i)$: evidence of receipt from R_i

---

[9] There might be a potential attack [Gürgens and Rudolph 2002] when the label l is constructed as $h(m, k)$ in the early literature, so this label is unique in each run and verifiable by any party. Note that the labels can be computed off-line.

Protocols	App.	Properties	Advantages	Drawbacks	Topology
OZCL'03	MPNR	different messages for different recipients	privacy, lightweight TTP	on-line TTP	1-N
OZL'04	MPNR	optimistic, different messages for different recipients	privacy, transparent TTP, asynchronous timeliness, efficient		1-N
ZOL'05	CEM	optimistic, no split of message in delivery	asynchronous timeliness, very efficient, recipient collusion resistance	more TTP overhead when TTP involved	1-N
ZOL'06	MPCS	optimistic, threshold cancel for weak asynchronous timeliness	abuse free, very efficient, 3 rounds and $3(n-1)$ steps		in-and-out

Table II. Our solutions' properties summary

- $Sub_k = S_O(fsub, R', L', t, E_{R'}(k))$: evidence of submission of the key to the TTP
- $Con_k = S_{TTP}(fcon, O, R', L', t, E_{R'}(k))$: evidence of confirmation of the key by the TTP

In this extension, the use of the same key for all users creates a new problem that did not appear in Kremer's online multi-party non-repudiation protocol. Because messages are different, when the same key k is used for encryption, and after the key k is published, any recipient will be able to read the messages addressed to the other recipients (by eavesdropping the messages that are transmitted between O and R). This problem is solved in this extended multi-party non-repudiation protocol, introducing some extra cost for the extended functionality over [Kremer and Markowitch 2000a].

3.1.1 *Protocol.* Here, we describe the protocol (see figure 4, where a dotted line indicates a *fetch* operation).

1. $O \rightarrow R_i$: $feoo, R_i, TTP, l_i, h(k), t, u_{R_i}, v_i, c_i, EOO_i$ for each $R_i \in R$
2. $R_i \rightarrow O$: $feor, O, l_i, EOR_i$ where $R_i \in R$
3. $O \rightarrow TTP$: $fsub, R', L', t, E_{R'}(k), Sub_k$
4. $O \leftrightarrow TTP$: $fcon, O, R', L', E_{R'}(k), Con_k$
5. $R'_i \leftrightarrow TTP$: $fcon, O, R', L', E_{R'}(k), Con_k$ where $R'_i \in R'$

The protocol works in the following way.

Step 1: O sends to every R_i evidence of origin corresponding to the encrypted message c_i, together with v_i. In this way, O distributes $|R|$ messages in a batch operation and each R_i gets the encrypted message as well as n_i. O selects the

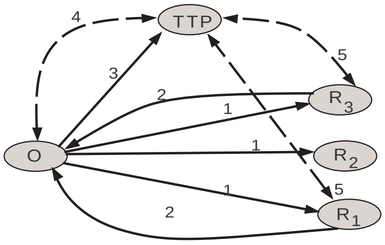

Fig. 4. Protocol with different messages

intended public key u_{R_i} being used in the encryption of n_i. If R_i disagrees (i.e., its authentication digital certificate has expired or been revoked), it should stop the protocol at this step. There is no breach of *fairness* if the protocol stops at step 1 because c_i cannot be obtained without key k, and it cannot be derived from message 1.

Step 2: Some entities (or all of them) send evidence of receipt of c_i back to O after checking evidence and labels. Again, there is no breach of fairness if the protocol stops.

Step 3: O sends k and Sub_k to the TTP in exchange for Con_k. The key k is encrypted using a group encryption scheme where the group of users is R'. Hence, only those entities belonging to R' will be able to decrypt and extract the key. Alike, O will obtain evidence only for the recipients included in the set R' that is submitted to the TTP. Note that, in this way, O can exclude some recipients which replied, but fairness is maintained.

Before confirming the key, the TTP checks that:

- $|R'| = |L'|$ holds
- current time $< t$

Step 4: O fetches Con_k from the TTP and saves it as evidence to prove that k is available to R'.

Step 5: Each R_i fetches $E_{R'}(k)$ and Con_k from the TTP. They will obtain k_i by computing k xor n_i. Also, they save Con_k as evidence to prove that k originated from O.

At the end of the protocol, if successful, the participants get the following evidence:

- NRO for honest recipient R_i: EOO_i, Con_k
- NRR for originator O: EOR_i, Con_k for all honest recipients $i \in R'$

Even though O can send a different deadline t, this is not an interesting option for it. This behavior will not affect fairness, since Con_k will not match with the rest of evidence obtained; i.e., no party will obtain valid evidence and the recipients could learn the message.

3.1.2 *Dispute Resolution.* Two kinds of disputes can arise: repudiation of origin and repudiation of receipt. Repudiation of origin arises when a recipient R_i claims having received a message M_i from an originator O who denies having sent it. Repudiation of receipt arises when the originator O claims having sent a message M_i to a recipient R_i who denies having received it.

Repudiation of Origin. If O denies sending M_i, R_i can present evidence EOO_i and Con_k plus $(TTP, t, u_{R_i}, v_i, c_i, n_i, k, E_{R'}(k), M_i, R', L')$ to the arbitrator. The arbitrator will check:

- $v_i = E_{u_{R_i}}(n_i)$
- $k_i = k$ xor n_i
- $c_i = E_{k_i}(M_i)$
- $l_i = h(O, R_i, TTP, h(c_i), h(k)) \wedge l_i \in L'$
- O's signature EOO_i
- TTP's signature Con_k

Repudiation of Receipt. If R_i denies receiving M_i, O can present evidence EOR_i and Con_k plus $(TTP, t, u_{R_i}, v_i, c_i, n_i, k, E_{R'}(k), M_i, R', L')$ to the arbitrator. The arbitrator will check:

- $R_i \in R'$
- $v_i = E_{u_{R_i}}(n_i)$
- $k_i = k$ xor n_i
- $c_i = E_{k_i}(M_i)$
- $l_i = h(O, R_i, TTP, h(c_i), h(k)) \wedge l_i \in L'$
- R_i's signature EOR_i
- TTP's signature Con_k

3.1.3 *Efficiency.* This protocol uses an on-line TTP which intervenes in every execution in a light-weight manner. We compare this approach with the one where an n-instance of a two-party protocol [Zhou and Gollmann 1996] is used in order to send messages to the intended parties. The efficiency of the three principal entities participating in the protocol is analyzed, using an operation comparison. For this comparison we will use the following basic operations:

- signature generation and verification
- generation of random numbers
- asymmetric encryption and decryption
- store and fetch operation

Depending on which algorithm is chosen for each of these operations, the bit complexity (as well as the bandwidth requirements) of each of the participating entities will change, although the relation going between them remains. Furthermore, the efficiency analysis also depends on how the group encryption primitive is used along the description of the protocol implemented. We will assume the complexity for a group encryption for n parties is equivalent to the cost of n asymmetric encryption operations. We denote:

$|R| = N$

$|R'| = N'$ (with $N' \leq N$)

$\approx$ roughly equal

$>$ or $<$ greater or smaller

$\gg$ or $\ll$ much greater or smaller

n-instanced two-party		**New approach**
Evidence of origin EOO_i	$=$	**EOO_i**
N signatures		N signatures
Generation of k_i	$\approx$	**Generation of n_i plus k**
Evidence of submission Sub_{k_i}	$\gg$	**Sub_k**
N' signatures		1 signature
Encrypted key $E_{u_{R_i}}(k_i)$	$\ll$	**Encrypted key $E_{R'}(k)$ plus $E_{u_{R_i}}(n_i)$**
N' asymmetric encryptions		N'+N asymmetric encryptions
N fetches operations of Con_{k_i}	$\gg$	**One fetch operation of Con_k**

Table III. O's computation complexity

n-instanced two-party		**New approach**
Evidence of receipt EOR_i	$=$	**EOR_i**
Fetch k_i and Con_{k_i}	$=$	**Fetch k and Con_k**
Obtain k_i	$<$	**Obtain k plus n_i**
Decrypts $E_{u_{R_i}}(k_i)$		Decrypt $E_{u_{R_i}}(k)$ Decrypt $E_{u_{R_i}}(n_i)$

Table IV. $R_i's$ computation complexity

n-instanced two-party		**New approach**
Store N' keys	$\gg$	**Store only one key**
Generation of N' evidences Con_{k_i}	$\gg$	**Generation of only one evidence Con_k**

Table V. TTP's computation complexity

Hence we can see in table V the TTP's efficiency is improved when it is generalized to multiple entities. Since communicating entities will usually pay for the TTP services, a more *efficient* and cheaper TTP service is achieved. In addition, we can see in tables III and IV that O's efficiency is improved too, while R_i's is slightly increased. Furthermore, the asymmetric encryption of n_i can be prepared off-line but, on the other hand, the fetch operations must be performed during the protocol run.

3.2 An Optimistic MPNR Protocol for Exchange of Different Messages

There is an optimistic approach in non-repudiation protocols where the entities are likely to behave honestly, thus giving priority to the main protocol and running sub-protocols only in case that an exception arises. Here we present an optimistic multi-party non-repudiation protocol based on [Ferrer-Gomila et al. 2002] [10], and use the same solution described in the previous section for the privacy of different messages.

Some additional notation in this protocol description is as follows:

- $R'' = R - R'$: a subset of R (in plaintext) with which O wants to cancel the exchange
- $R''\_finished$: a subset of R'' that have finished the exchange with the finish sub-protocol
- $R''\_cancelled = R'' - R''\_finished$: a subset of R'' with which the exchange has been cancelled by the TTP
- $l = h(M_1, M_2, .., k)$: label [11] that identifies the protocol run computed as the hash outcome of the concatenation of every encrypted message plus the key k
- $k_T = E_{u_{TTP}}(k)$: key k encrypted with the TTP's public key
- $EOO_i = S_O(feoo, R_i, TTP, k_T, l, v_i, u_{R_i}, h(c_i))$: evidence of origin for R_i
- $EOR_i = S_{R_i}(feor, O, TTP, k_T, l, v_i, u_{R_i}, h(c_i))$: evidence of receipt from each R_i
- $Sub_k = S_O(fsub, R', l, k)$: evidence of submission of the key to recipients
- $Cancel_{req} = S_O(TTP, R'', l)$: evidence of request of cancellation issued by the originator to the TTP
- $Cancel_O = S_{TTP}(O, l, R'', R''\_cancelled, Cancel_{req})$: evidence of cancellation issued by the TTP to the originator
- $Cancel_{R_i} = S_{TTP}(R_i, l, EOR_i, Cancel_{req})$: evidence of cancellation issued by the TTP to R_i
- $Con_k = S_{TTP}(R_i, l, k)$: confirmation evidence of k issued by the TTP

3.2.1 *Protocol.* The protocol consists of a *main* protocol (which will be the only one executed by the entities in the normal situation) and two sub-protocols: *cancel* and *finish* (see figure 5). The TTP is only involved in the sub-protocols in case of any participant's misbehavior or channel failure between the originator and the recipients. Any participant can initiate the corresponding sub-protocols to terminate a protocol run at any time without loss of fairness.

The *main* protocol executed by the final entities is:

1. $O \rightarrow R_i \;:\; feoo, R_i, TTP, k_T, l, v_i, u_{R_i}, c_i, EOO_i$ for each $R_i \in R$
2. $R_i \rightarrow O \;:\; feor, O, l, EOR_i$ where $R_i \in R$
3. $O \Rightarrow R' \;:\; fsub, l, E_{R'}(k), Sub_k$

---

[10] The protocol [Ferrer-Gomila et al. 2002] has some security errors as being identified in [Zhou 2004]. Those problems have been corrected in the design of this new protocol.

[11] Note that with the reduction to 3 steps in the main protocol, the attack proposed in [Gürgens and Rudolph 2002] does not work in this approach.

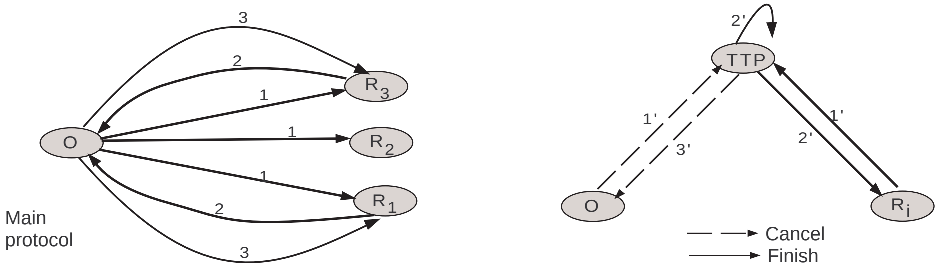

Fig. 5. Optimistic protocol with different messages

In step 1, the originator sends to each recipient its message encrypted with k_i. A recipient can derive k_i from k and a random number n_i which are also sent in this step (in a confidential way). Note that the TTP is included in this step, thus there is no confusion about which TTP to use in case they have to launch any of the sub-protocols. The originator picks each receiver's public key. If any recipient does not want to use such a key (e.g., the correspondent public key certificate has been revoked), then it stops the protocol. Otherwise, after verifying the data obtained, the recipient sends to the originator evidence of receipt at step 2, and the originator sends to the set of recipients who replied after a reasonable amount of time at step 3, the key and evidence of submission of that key, as a second part of evidence of origin.

If O did not receive a correct message 2 from some of the recipients R'', O may initiate the following *cancel* sub-protocol:

```
$1'.\ O \rightarrow TTP$ : $TTP, R'', l, Cancel_{req}$
$2'.\ TTP$                 FOR (all $R_i \in R''$)
                              IF ($R_i \in R''\_finished$) THEN retrieves $EOR_i$
                              ELSE appends $R_i$ into $R''\_cancelled$
$3'.\ TTP \rightarrow O$ : <all retrieved $EOR_i$ >, $R''\_cancelled$, $Cancel_O$
```

In this case, the originator communicates the TTP its intention of revoking the protocol with entities contained in R'' and for the protocol run labelled l. After verifying O's cancel request, the TTP checks which entities previously resolved the protocol and retrieves their proofs of receipt. The TTP generates evidence of cancellation for the rest of entities and includes everything in a message destined to the originator. No information about the message, keys and EOO is sent as [Ferrer-Gomila et al. 2002] does because a well-defined label for indexing purposes on the TTP side is used.

If some recipient R_i did not receive message 3, R_i may initiate the following *finish* sub-protocol:

$$\begin{array}{lll} 1'.\ R_i \rightarrow TTP & : & TTP, k_T, l, v_i, u_{R_i}, h(c_i), EOO_i, EOR_i \\ 2'.\ TTP \rightarrow R_i & : & \text{IF } (R_i \in R''\_cancelled) \text{ THEN } R_i, l, R'', Cancel_{req}, Cancel_{R_i} \\ & & \text{ELSE } \{R_i, l, E_{u_{R_i}}(k), Con_k \text{ AND} \\ & & \text{appends } R_i \text{ into } R''\_finished \text{ and stores } EOR_i\} \end{array}$$

The recipient sends to the TTP all the information that it has already got from the originator along with its evidence of receipt. If this entity does not belong to the group of entities with which the originator has cancelled the exchange, the TTP verifies all the information (digital signatures) and decrypts k_T, obtaining the key for the recipient. It also stores EOR_i. Note that if the protocol has been cancelled, it should be impossible for the recipient to cheat the TTP in a way that the TTP reveals the key k for that protocol run. For such a reason, the TTP must verify O's signature in the first step and check that l and k_T provided by the recipient fits with the information contained in EOO_i. This becomes above all important when confidentiality should be provided, since it prevents using the TTP as a decryption oracle (e.g., a recipient sending in other execution message $1'$ trying to get k_T decrypted). It also avoids replay attacks as discovered in some optimistic protocols [Shao et al. 2005].

Otherwise, the TTP sends a cancellation evidence to the recipient such that the latter can easily demonstrate to an arbitrator that the exchange was cancelled in case a dispute arises. This evidence includes the request of cancellation, such that the TTP's behavior is verifiable while the TTP need not to store all the request evidences from the originator.

Note that the originator has no interest in sending a subset $\bar{R}'' \neq R''$ when requesting to the TTP for cancellation. If $\bar{R}'' \supset R''$, then O will cancel the exchange with those $R_i \in \bar{R}'' - R''$ that have replied with EOR_i. If $\bar{R}'' \subset R''$, then $R_i \in R'' - \bar{R}''$ may invoke the *finish* sub-protocol to get the key k but O will not obtain EOR_i from the TTP at the above *cancel* sub-protocol.

At the end of the protocol, if successful, the participants get the following evidence:

- NRO for honest recipient i: EOO_i, Sub_k or Con_k
- NRR for originator O: EOR_i for all honest recipients $i \in R'$

3.2.2 *Dispute Resolution.* As we have mentioned, two kinds of disputes can arise. Here we further discuss the rules for their resolution.

Repudiation of Origin. If O denies sending M_i, R_i can present evidence EOO_i and Sub_k (or Con_k) plus $(TTP, l, u_{R_i}, v_i, c_i, n_i, k, k_T, M_i)$ to the arbitrator. The arbitrator will check:

- $v_i = E_{u_{R_i}}(n_i)$
- $k_i = k$ xor n_i
- $c_i = E_{k_i}(M_i)$
- O's signature on EOO_i
- O's signature on Sub_k, or TTP's signature on Con_k
- R_i is in the signed token Con_k or $R_i \in R'$ as signed in Sub_k

Repudiation of Receipt. If R_i denies receiving M_i, O can present evidence EOR_i plus $(TTP, l, u_{R_i}, v_i, c_i, n_i, k, k_T, M_i)$ and $(R'', R''\_cancelled, Cancel_O)$ if it has. The arbitrator will check:

- $v_i = E_{u_{R_i}}(n_i)$
- $k_i = k$ xor n_i
- $c_i = E_{k_i}(M_i)$
- R_i's signature on EOR_i
- (TTP's signature on $Cancel_O$) $\wedge$ $(R_i \notin R''\_cancelled)$

O will win the dispute if all the above checks are positive. If all the checks, but the last, are positive and O cannot present evidence $Cancel_O$, the arbitrator must further interrogate R_i. If the latter cannot present $Cancel_{R_i}$ (or this token is not properly constructed including $Cancel_{req}$ from O), O also wins the dispute. Otherwise, R_i can repudiate having received the message M_i.

We can also see that evidence provided by the TTP is *self-contained*, that is, the TTP need not to be contacted in case a dispute arises regarding the occurrence or not of the *cancel* sub-protocol launched by O. Thus, the TTP is efficiently verifiable. For instance, if the TTP cheats and distributes evidence of cancellation to R_i when O did not cancel the protocol for it, TTP's misbehavior will be visible (either the cancellation token will not be properly constructed with O's evidence of request embedded or it will reveal TTP's misconduct since $R_i \notin R''$). Additionally, it is important to know that the adjudicator does not need to check that $k_T = E_{u_{TTP}}(k)$ holds.

3.2.3 *Efficiency.* This protocol is very efficient in the case of a good behavior of the participating entities. In fact, three steps is the minimum number of steps that could be reached without breaking fairness in non-repudiation protocols. Even with multiple recipients for exchange of different messages, it manages to use only one key for evidence distribution, thus decreasing the computation and verification requirements for the originator and the TTP. For this new feature, public key encryption and decryption of temporal random numbers are the main extra cost added.

It is straightforward to see that this protocol is more efficient than any combination of two-party protocols, since it permits to send different messages in a confidential way to multiple entities as well as to cancel the protocol for a group of entities R'' in only one run of the *cancel* sub-protocol.

In addition, this protocol achieves asynchronous timeliness, because each entity can terminate, if needed, the protocol at any time at their own discretion while maintaining fairness. Nevertheless, asynchronous timeliness, though desirable for users, augments the requirements on the communication channel between the TTP and users. With the model assumed, the TTP could setup an evidence server (e.g., FTP server) such that users participating in the protocol can access (only with the read permission) and retrieve evidence. By *policy*, there must exist a deadline after which evidence cannot be retrieved. (This deadline can match evidence validity depending on the TTP's storage capacity.) With this transformation the protocol is adapted with respect to timeliness in a similar way as the on-line version is.

It seems to us very important to compare this off-line solution with that proposed by Markowitch et. al [Markowitch and Kremer 2000], even though the latter one did not address multiple messages. On the other hand, this is the only one we can compare with as an off-line multi-party non-repudiation protocol. Since the protocol has already been briefly described in the previous section, we present here several tables to make the comparison straightforward. For efficiency comparisons, the following measurement units have been taken into account [12]

- *Public key operations* (containing encryption and decryption as well as digital signature and verification).
- Number of items in the *storage*.
- Number of *steps* as the number of messages which need to be sent over the network.
- P as the number of recipients which need to contact the TTP for help and succeed.
- $N = |R|$ as the number of participating recipients.

	New approach	**Markowitch's**
1.Number of steps in main protocol	3	4
2.Number of steps in sub-protocol	2 (finish) 2 (cancel)	2 (if recovered or early) 4 (if not recovered yet)
3.Timeliness	Async	Sync
4.Items to be stored in the TTP	P+1	1
5.Additional interrogation needed in the dispute resolution process	Yes	No
6.TTP network access (reliability in the inverse direction)	No	Yes
7.Risk of unfairness	No	Yes (O must stop main protocol)
8.Transparency	Yes	No

Table VI. General off-line solution comparison

In table VI, we can observe that generally and when possible, this protocol improves Markowitch's version. Note that, for instance, it is not possible to reduce the number of items the TTP needs to store as a consequence of being different messages and, thus, different evidence of receipt (since the transparency property for O is maintained). This storage capacity request could be easily avoided by allowing the TTP to issue an affidavit for the finished entities, but then, transparency would be lost. There is an improvement in the number of steps needed in the main protocol, which fortunately will be the most frequently used, but the cost is the need of rarely having to interrogate an additional entity in case of disputes.

---

[12] When several actions are possible, we consider the worst case.

	New approach	**Markowitch's**
1.O's main protocol	4+2P+N	4+2P
2.R_i's main protocol	3	4
3.O's cancel/recovery protocol	2+P (cancel)	1 (recovery)
4.R_i's finish protocol	2	2
5.TTP's cancel protocol	2 (once)	-
6.TTP's finish protocol	5	5

Table VII. Public key operations comparison

In table VII, we can observe how, in general, this protocol seems to perform worse. But if we analyze the table, we will discover that this is due to the new functionality introduced and thus, unavoidable. Specifically, operations of O's main protocol are augmented in N because the number of different messages to be sent. For the same reason mentioned in previous paragraph, O needs to verify P different evidence of receipt from honest entities which contacted the TTP for finishing the protocol on time. For the rest of steps, both protocols perform in a similar way.

3.3 Fairness vs. Collusion

As it has been introduced in this paper, there exist different levels of fairness. This protocol provides two different levels depending on the initial assumptions and thus, depending on these assumptions the applications which will make use of them vary.

In the description of the previous protocols we assume that no collusion is possible between recipients. This will preserve a strong fairness property and applications benefited of these kind of protocols can be of any type.

Nevertheless, R_i could get c_i in the initial steps of the protocol and quit. Then, colluding with any other party and getting the unique key k, it could decrypt c_i without providing any evidence of receipt. Note that this issue is very difficult to solve as it is not possible to prevent one (semi) honest entity from sharing a secret once it decrypts it. However, it is also important to note that when a recipient misbehaves and colludes with a (semi) honest recipient in order to get the key and thus (only) the message intended to him, it will not in any case obtain evidence of origin, unless the originator obtains evidence of receipt. This is what has been defined as light fairness.

There are multiple applications which only need light fairness in their transactions. A common feature in these applications is that it is more important the exchange of evidence than the message content itself. That is, it is critical that both or none of the entities obtain evidence. For instance, some notification systems only need light fairness. This happens when the message itself is known by the recipient (e.g. birthday certificate) but in the notification, the receiver wants to be able to demonstrate to a third party its origin (e.g. official administration office) and the originator wants to be able to demonstrate if necessary that the message was indeed delivered to the intended receiver.

4. APPLICATION: EXTENSIONS TO OPTIMISTIC FAIR CEM

In this section we analyze and provide extensions to a fast and simple optimistic fair CEM protocol [Micali 2003] for achieving timeliness and multi-party fair exchange [13]. Firstly, we briefly describe Micali's optimistic protocol for CEM.

4.1 A Protocol for Fair CEM with Deadline Time

Although Micali explained different fair CEM protocols in [Micali 2003], we describe here the most complete one that supports confidentiality, fairness and synchronous timeliness. Because it is an optimistic protocol, if both parties behave honestly, the TTP (which is also referred as the post office PO here) will not be involved. Each user in the system has a unique identifier. Before sending the plaintext message M to the recipient, the originator computes a secret Z protected with the TTP's encryption key as $Z = E_{PO}(O, R, E_R(M))$. To reach timeliness, Micali proposed a deadline solution, where originator chooses a time t after which the TTP should not help the recipient in the conclusion of the protocol.

$$\begin{aligned} &1.\ O \rightarrow R \ :\ t, Z, S_O(t, Z) \\ &2.\ R \rightarrow O \ :\ S_R(Z) \\ &3.\ O \rightarrow R \ :\ E_R(M) \end{aligned}$$

Whenever R reaches step 1 and verifies O's signature, it must extract the deadline t and estimate whether it will have enough time to contact the PO in case of O's misbehavior or channel failure. Variable t_D denotes the maximum possible time discrepancy that R believes to exist between his clock and that of the PO. If R receives step 1 in time t_R (i.e., recipient's local time) such that $t_R + t_D$ is greater than or equal to t, then R halts; otherwise it proceeds to step 2. After verifying R's signature, O sends the message M to the recipient at step 3.

After replying at step 2, if the recipient does not get the message within a reasonable amount of time, or $Z = E_{PO}(O, R, E_R(M))$ does not hold, R contacts the PO in a *resolve* sub-protocol:

$$\begin{array}{ll} 1'.\ R \rightarrow PO & :\ t, Z, S_O(t, Z), S_R(Z) \\ 2'.\ \text{IF } (t_{PO} < t) \text{ AND (valid signatures)} & \\ \qquad PO \rightarrow R & :\ X \\ \qquad PO \rightarrow O & :\ S_R(Z) \end{array}$$

In this sub-protocol, the PO verifies whether R's request arrives before O's deadline and also whether both signatures are correct. If so, the PO decrypts Z with its private key and, if the result is a triplet consisting of O, R, and an unknown string X, it sends X to the recipient and forwards R's signature to the originator.

Note that the semantics of the deadline t is different from the previous solutions where clock shifts between the entities could not lead to a security problem, but to a timeliness problem. Even though the deadline parameter is used for finalization of the protocol and consequently for timeliness, the meaning is different due to the optimistic approach. When the TTP participates in an on-line manner, the deadline establishes a point after which the protocol is resolved and the recipients can contact the TTP. In this case, the deadline indicates the point before which

---

[13] The work in this section has been partially published in [Zhou et al. 2005].

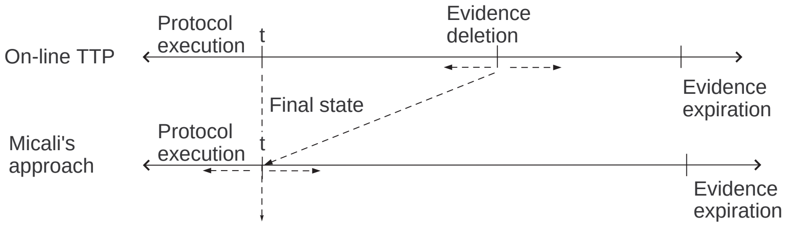

Fig. 6. Deadlines in fair CEM

the recipients must contact the TTP if needed, because afterwards, it will not help (and thus fairness will be damaged).

It can be argued that, at the end, both solutions need a point of time in which the TTP cannot help any more and if recipients do not access in advance, fairness will be also damaged. Nevertheless, there is a very important difference in both approaches which will be perhaps better appreciated in figure 6.

When evidence deletion occurs at the TTP, the latter will not be able to help entities any more. This event can coincide with evidence expiration or not depending on the TTP's storage resources. Nevertheless, time (deadline *t*) at which the state is final (synchronous timeliness) and evidence deletion are different. In this way, entities can know beforehand when the protocol has a definitive state and then, if needed, seek the TTP's help. On the other hand, Micali's solution unifies deadline and evidence deletion. [14] If this deadline is selected too early, mail recipients could not have enough time to resolve the protocol. If greater, entities will need to wait in order to know the final state of the protocol, perhaps more than what is desired.

4.2 Extension to Fair CEM with Asynchronous Timeliness

Here, a different solution for timeliness, *asynchronous timeliness* (i.e., either party can finish the protocol at any time without loss of fairness) is proposed. In Micali's proposal, even if the recipient approximately calculates in each run the time to contact the PO, there can be always a situation in which the PO is unaccessible for longer. In such a case, the recipient will not get the item from the (un-)fair exchange protocol and it will be difficult to figure out who bears the responsibility for the breach of fairness.

With the *resolve* sub-protocol, the recipient will be able to finish the protocol at any time. Similarly, if O does not want to wait for the resolution of the protocol it can abort it with *cancel* sub-protocol at any time too. The revised *main* protocol, in which O's signature is not needed anymore [15] in step 1 due to the omission of the deadline, is as follows:

$$\begin{array}{l} 1.\ O \rightarrow R\ :\ Z \\ 2.\ R \rightarrow O\ :\ S_R(Z) \\ 3.\ O \rightarrow R\ :\ E_R(M) \end{array}$$

---

[14] There is no deletion event, but the meaning is the same; i.e., the TTP's help is not available.

[15] In CEM applications, only non-repudiation of receipt is mandatory.

The revised *resolve* sub-protocol which will be requested by the recipient under the same conditions as the original one is as follows:

$$\begin{aligned} 1'.\ R \rightarrow PO &: Z, S_R(Z) \\ 2'.\ R \leftrightarrow PO &: \text{IF cancelled THEN}\ \ S_O(cancel, Z) \\ &\ \ \text{ELSE}\ \ \ E_R(M) \end{aligned}$$

When the PO receives such a request, it first checks R's signature on Z. If valid, the PO further decrypts Z and extracts the identities of sender and recipient of Z. If R is the intended recipient of Z and the exchange has not been cancelled by O, the PO marks the exchange status related to $(O, R, h(Z))$ as resolved, sends $E_R(M)$ to R, and stores $S_R(Z)$ (which will be collected by O when it initiates the *cancel* sub-protocol). If the exchange has been cancelled by O, the PO forwards $S_O(cancel, Z)$ to R, and R can use this evidence to prove that O has cancelled the exchange.

The new *cancel* sub-protocol is as follows:

$$\begin{aligned} 1'.\ O \rightarrow PO &: Z, E_{PO}(S_O(cancel, Z)) \\ 2'.\ O \leftrightarrow PO &: \text{IF resolved THEN}\ \ S_R(Z) \\ &\ \ \text{ELSE}\ \ \ ack \end{aligned}$$

When the PO receives such a request from the originator, it first checks O's signature after decrypting with its corresponding asymmetric key [16]. If valid, the PO further decrypts Z and extracts the identities of sender and recipient. If O is the sender of Z and the exchange status related to $(O, R, h(Z))$ is marked as resolved, the PO forwards $S_R(Z)$ to O. Otherwise, the PO marks the exchange status related to $(O, R, h(Z))$ as cancelled, and acknowledges O's cancelation.

Because there is no deadline for the recipient, the originator can abort the protocol as desired. As we can see, the PO provides access to the items that any of the parties needs. The parties can access the data at any time and it is not the responsibility of the PO whether the users get the messages they expect. The PO is not stateless and needs to maintain (within a reasonable amount of time) a table with entries $(O, R, h(Z), state)$ and to store the signatures for serving the users. Therefore, the server implementing the PO must secure this information (although confidentiality is not needed).

In case of a dispute, both parties must agree which arbitrator will evaluate the final outcome of the protocol based on the evidence provided by the users. Consequently, if the recipient denies having received a message M in a CEM protocol run, the originator should provide $(Z, S_R(Z))$ and the arbitrator settles that the originator sent the message M to the recipient if:

- $Z = E_{PO}(O, R, E_R(M))$ holds;
- the recipient's signature on Z is valid;
- the recipient cannot provide $S_O(cancel, Z)$.

---

[16] As pointed out in [González-Deleito 2005], O needs to encrypt its own signature in order to avoid an attack which allows the recipient to get the cancel token (eavesdropping O's request from message 1′) when the protocol has been previously resolved by the recipient (and thus making the originator believe the exchange succeeded).

It is easy to notice that integrity of the message is not ensured. Any external attacker (or R itself) can create a message $Z' = E_{PO}(O, R, E_R(M'))$. In this case, the originator will obtain a proof for a message it did not send and will try to cancel the protocol. If it is not possible (R already resolved the protocol), the recipient will have a message the originator did not send but cannot demonstrate it (the process is anonymous because there are no digital signatures). So this protocol does not ensure the service, but it does ensure that the recipient can read the message the originator sent if and only if the originator obtains evidence of receipt. If integrity and authenticity of the message for ensuring the service are properties needed by the originator, then it needs to digitally sign Z at step 1 and the receiver needs to provide O's signature at step 1' of the resolve sub-protocol for PO's verification.

4.3 Extension to Multi-Party Fair CEM

There are scenarios in which the participation of multiple entities can result in an important improvement. CEM is one of them. We can easily figure out applications in which sending e-mail to several users is feasible and useful. In section 2 several solutions for multi-party CEM were pointed out. Based on Micali's CEM protocol, this extension provides a new feature such that the sender can distribute in a certified manner a message M to several recipients. Some additional notation in the protocol description is:

- R : a set of intended recipients.
- $Header$: a header indicating in which position the recipient has to look for its information (e.g. $[R_i, j]$).
- M : message being sent from the originator to the recipients R.
- $R'' = R - R'$: a subset of R (in plaintext) with which the originator wants to cancel the exchange.
- $R''\_resolved$: a subset of R'' that have resolved the exchange with the *resolve* sub-protocol.
- $R''\_cancelled = R'' - R''\_resolved$: a subset of R'' with which the exchange has been cancelled by the PO.
- $u_R = u_{R_1}, u_{R_2}, ...$: concatenation of public keys from group R.
- $E_R(M)$: an group encryption of M for group R.
- $Z = E_{PO}(O, R, E_R(M))$: a secret Z protected with the PO's encryption key.

The PO will participate, if requested by any entity, in a mutually exclusive way (i.e., atomic execution of the sub-protocols for each user). Here, we describe the protocol.

The *main* protocol executed by the final entities is:

$$\begin{array}{ll} 1.\ O \Rightarrow R & :\ Header, u_R, Z \\ 2.\ R_i \rightarrow O & :\ S_{R_i}(u_{R_i}, Z) \text{ where each } R_i \in R \\ 3.\ O \Rightarrow R' & :\ E_{R'}(M) \end{array}$$

In step 1, the originator sends the secret Z and all the recipients' public keys such that if any recipient does not agree with its public encryption key (e.g., the corresponding public key certificate has been revoked), then it stops the protocol.

Otherwise, after verifying the data obtained, the recipient sends evidence of receipt to the originator at step 2, and the latter sends the (encrypted) message M at step 3 to the set of recipients who replied.

If the originator did not receive a correct message 2 from some of the recipients R'', she may initiate the following *cancel* sub-protocol:

$$\begin{array}{lll} 1'. \ O \rightarrow PO & : & Z, R'', S_O(cancel, R'', Z) \\ 2'. \ PO & & \text{FOR (all } R_i \in R''\text{)} \\ & & \text{IF } (R_i \in R''\_resolved) \text{ THEN retrieves } S_{R_i}(u_{R_i}, Z) \\ & & \text{ELSE appends } R_i \text{ into } R''\_cancelled \\ 3'. \ O \leftrightarrow PO & : & \text{all retrieved } S_{R_i}(u_{R_i}, Z),\ S_{PO}(R''\_cancelled, Z) \end{array}$$

In this case, the originator communicates to the PO its intention of revoking the protocol with entities contained in R''. After verifying the originator's cancel request, the PO checks which entities previously resolved the protocol and gets their proofs of receipt. Then, the PO generates evidence of cancelation for the rest of entities and includes everything in a message destined to the originator.

If some recipient R_i did not receive the message 3 or it was not valid, R_i may initiate the following *resolve* sub-protocol:

$$\begin{array}{lll} 1'. \ R_i \rightarrow PO & : & Header, u_{R_i}, Z, S_{R_i}(u_{R_i}, Z) \\ 2'. \ R_i \leftrightarrow PO & : & \text{IF } (R_i \in R''\_cancelled) \text{ THEN } S_{PO}(R''\_cancelled, Z) \\ & & \text{ELSE } \{E_{R_i}(M) \\ & & \text{appends } R_i \text{ into } R''\_resolved \text{ and stores } S_{R_i}(u_{R_i}, Z)\} \end{array}$$

The recipient sends to the PO all the information that it has already got from the originator along with its evidence of receipt. If this entity does not belong to the group of entities with which the originator cancelled the exchange, the PO verifies all the information (digital signatures) and decrypts Z. It also stores $S_{R_i}(u_{R_i}, Z)$. Note that if the protocol has been cancelled, it should be impossible for the recipient to cheat the PO in a way that the PO reveals the message for that protocol run. For that reason, the PO must verify the recipient's signature as well as integrity of Z in the first step.

Note that the recipient can cheat the TTP by manipulating the *Header* field. Nevertheless, this will not disclose any other message, since it is encrypted with the intended recipient's public key. Furthermore, this action is nonsense due to two reasons: (i) message M is similar for every recipient regardless of the position in $E_R(M)$ and (ii) R_i will bring itself to an unfair situation since it will not be able to get the message while the originator can obtain from the TTP its receipt signature.

If this entity belongs to the group of entities with which the originator cancelled the exchange, the PO sends a cancelation evidence to the recipient such that the latter can easily demonstrate to an arbitrator that the exchange was cancelled in case a dispute arises.

If R_i denies having received M, the originator can present evidence $Z, E_R(M)$, $S_{R_i}(u_{R_i}, Z), S_{PO}(R''\_cancelled, Z)$ and the arbitrator settles that the recipient received the message M from the originator if:

- R_i can decrypt $E_R(M)$ with u_{R_i} and the outcome is M;
- $Z = E_{PO}(O, R, E_R(M))$ holds;

- R_i's signature on Z and its encryption public key is valid;
- PO's signature on $S_{PO}(R''\_cancelled, Z) \wedge R_i \notin R''\_cancelled$.

The originator will succeed on the dispute if all the above checks are positive. If all the checks but the last are positive and it cannot present evidence of cancelation, then the arbitrator must further interrogate R_i. If the latter cannot present $S_{PO}(R''\_cancelled, Z)$ in which $R_i \in R''\_cancelled$, the originator also wins the dispute. Otherwise, R_i can repudiate having received the message M. Therefore, evidence provided by the PO is *self-contained*, that is, the PO need not be contacted in case a dispute arises regarding the occurrence of a cancelation sub-protocol launched by the originator.

5. APPLICATION: MULTI-PARTY CONTRACT SIGNING

In this section, we explain a new synchronous multi-party contract signing protocol that, with n parties, it reaches a lower bound of $3(n-1)$ steps in the all-honest case and $4n-2$ steps in the worst case (i.e., all parties contact the TTP). This is so far the most efficient synchronous multi-party contract signing protocol in terms of the number of messages required. It also considers additional features like timeliness and abuse-freeness in an improved version [17].

In all practical schemes, contract signing involves a TTP which plays the role of a notary in paper-based contract signing and somehow shares the legal duties the former ones have. In fact, designing and implementing a contract signing protocol using an in-line TTP should not be a complicated task. In this case, if Alice and Bob wish to enter into a contract, they each sign a copy of the contract and send it to the TTP through a secure channel. The TTP will forward the signed contracts only when it has received valid signatures from both Alice and Bob.

In our continuous search for speeding up our daily life activities, it is desirable not using a TTP in a contract signing protocol. Additionally, if the TTP is not involved, the notary fee could be avoided. Some protocols appear in the literature trying to eliminate the TTP's involvement using *gradual or probabilistic exchange* of signatures [Blum 1981; Even et al. 1985]. In [Even et al. 1985], Even et al. propose a contract signing protocol which makes use of *oblivious transfer*. This allows the transfer of a recognizable (e.g. signed) message M such that the recipient can read it with a probability 1/2 while the originator has no way of knowing whether the recipient could read it or not. With this tool, they provided a probabilistic contract signing protocol with a very high success (fairness) probability.

Nevertheless, and specially for contract signing protocols, a signer would not like to risk one million dollars when the deal is done. Therefore, these solutions may not be accepted by signatories. Furthermore, users always deal with the presence of TTPs when important contracts are to be signed (e.g., the act of selling a house in which usually a notary is involved) in the paper-based world.

Thus, in this section, we focus on deterministic optimistic contract signing protocols. A synchronous model is used, in which we assume messages sent among participants can be lost in the network, but a message from a participant reaches the TTP in a finite and known amount of time. Attackers can insert, delete and

---

[17] The work in this section has been partially published in [Zhou et al. 2006].

modify messages, but it is assumed that attackers cannot break the clock synchronization of the network and cannot forge digital signatures. Under this model, the number of rounds can be made independent of the number of participants.

5.1 Improved Synchronous MPCS Protocol

Here, we first present a simple synchronous protocol for multi-party contract signing. As in Asokan's approach, this is also based on two differentiated phases: a promise to sign, and a real signature that a party releases only after receiving all promises from the rest of participants. Again, in the same manner, this protocol reaches a lower bound of $4(n-1)$ steps in the all-honest case and $5n-3$ steps in the worst case when all parties contact the TTP. This result will be further improved in the optimal version by reducing the number of steps to $3(n-1)$ in the all-honest case and $4n-2$ in the worst case.

5.1.1 *A Simple Version.* Let us consider the following simple solution which uses verifiable encryption of signatures based on a ring architecture for achieving *transparency* of the TTP. Assume that the channel between any participant and the TTP is functional and not disrupted. The following notation is used in the protocol description.

- $C = [M, P, id, t]$: a contract text M to be signed by each party $P_i \in P(i = 1, \cdots, n)$, a unique identifier id for the protocol run, and a deadline t agreed by all parties to contact the TTP.
- $Cert_i$: a certificate with which anyone can verify that the ciphertext is the correct signature of the plaintext, and can be decrypted by the TTP (see CEMBS - *Certificate of an Encrypted Message Being a Signature* in [Bao et al. 1998]).

A simple linear protocol for multi-party contract signing is sketched as follows:

$$
\begin{array}{lll}
1. & P_1 \rightarrow P_2 & : m_1[= C, e_{TTP}(S_{P_1}(C)), Cert_1] \\
2. & P_2 \rightarrow P_3 & : m_1, m_2[= C, e_{TTP}(S_{P_2}(C)), Cert_2] \\
n-1. & P_{n-1} \rightarrow P_n & : m_1, .., m_{n-1}[= C, e_{TTP}(S_{P_{n-1}}(C)), Cert_{n-1}] \\
n. & P_n \rightarrow P_{n-1} & : m_n[= C, e_{TTP}(S_{P_n}(C)), Cert_n] \\
n+1. & P_{n-1} \rightarrow P_{n-2} & : m_{n-1}, m_n \\
2(n-1). & P_2 \rightarrow P_1 & : m_2, m_3, .., m_n \\
2n-1. & P_1 \rightarrow P_2 & : S_{P_1}(C) \\
2n. & P_2 \rightarrow P_3 & : S_{P_1}(C), S_{P_2}(C) \\
3(n-1). & P_{n-1} \rightarrow P_n & : S_{P_1}(C), S_{P_2}(C), .., S_{P_{n-1}}(C) \\
3n-2. & P_n \rightarrow P_{n-1} & : S_{P_n}(C) \\
3n-1. & P_{n-1} \rightarrow P_{n-2} & : S_{P_{n-1}}(C), S_{P_n}(C) \\
4(n-1). & P_2 \rightarrow P_1 & : S_{P_2}(C), S_{P_3}(C), .., S_{P_n}(C)
\end{array}
$$

The above main protocol is divided into two phases. The parties first exchange their commitments in an "in-and-out" manner. Note that P_1 can choose t in the first message (and others can halt if they do not agree). Only after the first phase is finished at step $2(n-1)$, the final signatures are exchanged. Following this simple approach, only $4(n-1)$ steps are needed.

If there is no exception (e.g., network failure or misbehaving party), the protocol will not need the TTP's help. Otherwise, the following resolve sub-protocol helps to drive the contract signing process to its end. P_i can contact the TTP before the deadline t.

```
1. $P_i \rightarrow TTP$ : $resolve_{P_i} = m_1, .., m_n$
2. $TTP$                 : IF $NOT\ resolved\ AND\ resolve_{P_i}$ is received before $t$ THEN
                              decrypts $m_1..m_n$
                              publishes $S_{P_1}(C), .., S_{P_n}(C)$
                              $resolved$=true
```

Boolean variable *resolved* is initialized to false. If the main protocol is not completed successfully, some parties may not hold all the commitments $(m_1, .., m_n)$. Then, they just wait until the deadline t and check with the TTP whether the contract has been resolved by other parties. If not, the contract is cancelled. Otherwise, they get the valid contract $(S_{P_1}(C), .., S_{P_n}(C))$ from the TTP.

If a party has all the commitments when the main protocol is terminated abnormally, it could initiate the above sub-protocol. Then the TTP will help to resolve the contract if the request is received before the deadline t, and the contract will be available to all the participants (even after the deadline t). After the deadline, the TTP will not accept such requests any more. In other words, the status of the contract will be determined the latest by the deadline t. Note that no party can cheat the TTP using a distinct deadline because in that case they will be cancelling or resolving a different contract. If parties do not want to advertise this data in the contract itself since timeout is not part of the text agreed, then the deadline can be somehow (hashed together with) included in the unique *id* which is the main variable used by the TTP to distinguish between different protocol instances as a contract reference. In this protocol, the TTP intervention is simple, elegant and lightweight.

In this case, the dispute resolution process is straightforward. If a party holds all the signatures, the contract is assumed to be valid.

5.1.2 *An Optimal Version.* The simple version protocol has two clearly differentiated phases: exchange of commitments and exchange of digital signatures. The number of steps can be further reduced if more available information is sent at each step and then both phases are merged. This will result in an improvement of the previous simple version protocol.

Using the same notation, an optimal synchronous protocol for multi-party contract signing is outlined as follows:

$1.\quad P_1 \rightarrow P_2 \quad : m_1[= C, e_{TTP}(S_{P_1}(C)), Cert_1]$
$2.\quad P_2 \rightarrow P_3 \quad : m_1, m_2[= C, e_{TTP}(S_{P_2}(C)), Cert_2]$
$n-1.\quad P_{n-1} \rightarrow P_n \quad : m_1, .., m_{n-1}[= C, e_{TTP}(S_{P_{n-1}}(C)), Cert_{n-1}]$
$n.\quad P_n \rightarrow P_{n-1} \quad : m_n[= C, e_{TTP}(S_{P_n}(C)), Cert_n], S_{P_n}(C)$
$n+1.\quad P_{n-1} \rightarrow P_{n-2} : m_{n-1}, m_n, S_{P_{n-1}}(C), S_{P_n}(C)$
$2(n-1).\quad P_2 \rightarrow P_1 \quad : m_2, m_3, .., m_n, S_{P_2}(C), S_{P_3}(C), .., S_{P_n}(C)$
$2n-1.\quad P_1 \rightarrow P_2 \quad : S_{P_1}(C)$
$2n.\quad P_2 \rightarrow P_3 \quad : S_{P_1}(C), S_{P_2}(C)$
$3(n-1).\quad P_{n-1} \rightarrow P_n \quad : S_{P_1}(C), S_{P_2}(C), .., S_{P_{n-1}}(C)$

The resolve sub-protocol used by participants to request the TTP's help does not change. Note that even though the two phases are merged, no party releases its plaintext signature of the contract without having first received all the commitments. If any party decides to quit before releasing its plaintext signature of the contract, the rest of participants can obtain all plaintext signatures of the contract with the TTP's help. As the protocol is similar to the previous one, the same requirements are fulfilled and the identical dispute resolution process is used by the adjudicator.

This optimal version allows overlapping the dispatch of promises with real signatures without loosing fairness. It improves the simple version presented in Section 5.1.1 by reducing the number of steps to $3(n-1)$ in the all-honest case and $4n-2$ in the worst case. Note that for $n = 2$, three messages are sufficient and optimal, as shown in [Pfitzmann et al. 1998].

5.2 Achieving Abuse-Freeness

The MPCS protocol presented in the previous section improved the lower bound of steps in existing synchronous MPCS protocols. However, it does not satisfy the properties of abuse-freeness and asynchronous timeliness. Here the protocol is further improved to address these properties.

Although it is not possible to force a participant to keep on following the steps of the protocol, the protocol can be designed in such a manner that it has no way to demonstrate to an outsider the contract is under its control. For this purpose, it uses a *blind commitment* that only the TTP can verify. With this concept of design in mind, the previous protocol can be modified to eliminate the "illustrative" information. The main protocol remains the same, but $Cert_i$ is not included in m_i. Instead, evidence of origin of the blind commitment $Commit_i$ is generated:

$$Commit_i = S_{P_i}(h(C), e_{TTP}(S_{P_i}(C)))$$

where $h(C)$ is the hash value of C to be used to establish a unique link between $Commit_i$ and C.

$$\begin{array}{lll}
1. & P_1 \rightarrow P_2 & : m_1[= C, e_{TTP}(S_{P_1}(C)), Commit_1] \\
2. & P_2 \rightarrow P_3 & : m_1, m_2[= C, e_{TTP}(S_{P_2}(C)), Commit_2] \\
n-1. & P_{n-1} \rightarrow P_n & : m_1, .., m_{n-1}[= C, e_{TTP}(S_{P_{n-1}}(C)), Commit_{n-1}] \\
n. & P_n \rightarrow P_{n-1} & : m_n[= C, e_{TTP}(S_{P_n}(C)), Commit_n], S_{P_n}(C) \\
n+1. & P_{n-1} \rightarrow P_{n-2} & : m_{n-1}, m_n, S_{P_{n-1}}(C), S_{P_n}(C) \\
2(n-1). & P_2 \rightarrow P_1 & : m_2, m_3, .., m_n, S_{P_2}(C), S_{P_3}(C), .., S_{P_n}(C) \\
2n-1. & P_1 \rightarrow P_2 & : S_{P_1}(C) \\
2n. & P_2 \rightarrow P_3 & : S_{P_1}(C), S_{P_2}(C) \\
3(n-1). & P_{n-1} \rightarrow P_n & : S_{P_1}(C), S_{P_2}(C), .., S_{P_{n-1}}(C)
\end{array}$$

Each party needs to check whether all the blind commitments it has received are valid before releasing its real signature of the contract. A *valid* blind commitment $Commit_i$ means it is from P_i (by checking its signature), linked to C (by checking $h(C)$), but does not guarantee that $e_{TTP}(S_{P_i}(C))$ in $Commit_i$ matches $S_{P_i}(C)$. $Commit_i$ is *correct* if it is valid and also matches $S_{P_i}(C)$.

If there is no exception (e.g., network failure or misbehaving party), the protocol will not need the TTP's help. Otherwise, a modified resolve sub-protocol helps to drive the contract signing process to its end. P_i can contact the TTP before the deadline t.

```
1. $P_i \rightarrow TTP$ : $resolve_{P_i} = m_1, .., m_n$
2. $TTP$              : IF $NOT\ resolved\ AND\ resolve_{P_i}$ is received before $t$
                        AND all $Commit_i$ are valid THEN
                          decrypts & verifies $m_1..m_n$
                          $resolved$=true
                          IF $S_{P_1}(C), .., S_{P_n}(C)$ ok THEN
                            publishes $S_{P_1}(C), .., S_{P_n}(C)$
                          ELSE IF $P_i \notin group_f$
                            publishes $fail, group_f, S_{TTP}(fail, C, group_f)$
```

Boolean variable *resolved* is initialized to false. When a party holding all the *valid* blind commitments initiates the above sub-protocol, the TTP will help to resolve the contract if the request is received before the deadline t. The TTP decrypts and verifies $m_1, \cdots, m_n$. If they are all correct, the TTP will publish $S_{P_1}(C), \cdots, S_{P_n}(C)$. Otherwise, the TTP will invalidate the contract by publishing a *fail* token $S_{TTP}(fail, C, group_f)$ where $group_f$ indicates the parties which misbehaved in generating their commitments.

The dispute resolution process is changed when the *fail* token is introduced. If a party can show this token, the contract is invalid. Therefore, with respect to the simple dispute resolution process defined previously, now the arbiter needs to interview both parties disputing the validity of the contract.

Furthermore, at the end of the main protocol, each party needs to check whether $e_{TTP}(S_{P_i}(C))$ in $Commit_i$ matches $S_{P_i}(C)$ for $i = 1, \cdots, n$ (assuming the encryption algorithm is deterministic). If not, it should initiate the above sub-protocol to get the *fail* token. A party P_i cannot get any advantage by providing different $Commit_i$ in the main protocol and the resolve sub-protocol. If P_i provides correct $Commit_i$ in the main protocol but incorrect $Commit'_i$ in the sub-protocol, P_i will not get the *fail* token, i.e., cannot cancel a protocol instance whose final state is signed. On the other hand, if P_i provides incorrect $Commit_i$ in the main protocol but correct $Commit'_i$ in the sub-protocol, P_i may get the signed contract if other parties did not misbehave in generating their commitments, but any other honest party can initiate the resolve sub-protocol to get the *fail* token, thus the contract is still invalid.

Remark 5.1. It seems that an external adversary could generate valid looking or even fake blind commitments himself. This would allow him to abort the protocol when desired, but evidence of origin avoid this situation.

The blind commitment does not allow a participant to demonstrate that the protocol state is under its control. In fact, in this case, getting all m_i does not mean being able to solve the protocol as in previous protocols presented in this section. Thus, it provides an abuse-freeness feature. Proof is straightforward, since there is no point in the protocol in which an entity can ensure, even to itself, that the contract is signed until plaintext signatures are obtained. The solution allows to maintain the same number of steps as the optimal protocol in Section 5.1.2. Furthermore, the TTP is still transparent in this sub-protocol because the signed contract published by the TTP is the same as the one obtained in the main protocol.

5.3 Achieving Timeliness

In the previous protocols just presented, a deadline t is selected by the first participant. If other participants disagree with the deadline, they can simply abort the execution of the protocol. Of course, this deadline could be negotiated among the participants before the contract signing protocol is initiated.

If the main protocol is not completed successfully, some participants may hold all the commitments while the others may only hold part of the commitments. For those holding all the commitments, they have the freedom to either resolve the contract with the TTP's help before the deadline t, or take no action and just let the contract being automatically cancelled after the deadline t.

As we have already mentioned, asynchronous timeliness is not fulfilled in these protocols, where a deadline t is used, forcing all the participants to approximately synchronize their clocks with the TTP's one. This task has been widely studied and standard solutions exist (such as the *network time protocol* [Mills 1992]). Nevertheless, asynchronous timeliness fits better with users' desires in MPCS protocols; since, for instance, contract conditions can also change with time, not being favorable any more for a group of entities.

For those only holding part of the commitments, they have no options but only wait until the deadline t to know the status of the contract. Obviously, this is unfavorable to these participants in term of timeliness. They should also have the right to decide the status of the contract before the deadline t. As they only hold part of commitments, they are not able to resolve the contract, so they can only choose to cancel the contract. (Note that in this "in-and-out" architecture of commitment exchange, for those participants only holding part of the commitments, even if all of them collaborate, their combined commitments are still incomplete to resolve the contract.)

Here we present a (j, n)-*threshold* cancel sub-protocol. As long as there are at least j out of n participants that wish to cancel the contract before the deadline t, the contract could be cancelled. The cancel sub-protocol is as follows, where *counter* (initial value equals to zero) records the number of cancel requests received by the TTP, and $group_c$ records the participants which made cancel requests. For simplicity of description, it is built based on the main protocol in Section 5.1.2 without considering abuse-freeness (but can be easily merged).

```
1. $P_i \rightarrow TTP$ : $cancel_{P_i} = C, cancel, S_{P_i}(C, cancel)$
2. $TTP$              : IF $cancel_{P_i}$ is received before $t$ AND $C$ is not resolved
                        AND $C$ is not cancelled THEN
                          stores $cancel_{P_i}$; $group_c = group_c + P_i$;
                          $counter++$;
                          IF $counter \geq j$ THEN
                            sets $C$ as cancelled
                            publishes $cancel, group_c, S_{TTP}(cancel, C, group_c)$
```

The resolve sub-protocol is modified as follows:

```
1. $P_i \rightarrow TTP$ : $resolve_{P_i} = C, m_1, .., m_n, S_{P_i}(C, m_1, .., m_n)$
2. $TTP$                 : IF $resolve_{P_i}$ is received before $t$ AND $C$ is not cancelled
                           AND C is not resolved
                               decrypts $m_1..m_n$
                               sets $C$ as resolved
                               publishes $S_{P_1}(C), .., S_{P_n}(C)$
```

With the above cancel and resolve sub-protocols, each participant has at least one option to determine the status of the contract before deadline t if the main protocol is not completed successfully. Thus timeliness is achieved, and the extent of timeliness depends on the threshold value j: strong timeliness when $j = 1$, and weak timeliness when $j = n$.

However, the threshold value j should be selected carefully. If j is too small, a few parties may collude to invalidate a contract. If j is too big, it might be hard to establish a valid cancel request among j parties. A possible option is $j = [n/2] + 1$, with a weak majority to "vote" for the validity of a contract.

In the dispute resolution, the *cancel* token issued by the TTP has the top priority. In other words, if a participant presents the *cancel* token, then the contract is invalid. That implies that if there are at least j out of n participants who want to cancel the contract before the deadline, even if they have released their plaintext signatures in the main protocol, they together can still change their mind before that deadline. This is a reasonable scenario in the real world because the situation defined in the contract may change with time, even during the process of contract signing, and each participant wishes to pursue the maximum benefit by taking appropriate actions (resolve or cancel).

As the *cancel* token from the TTP has higher priority than the signed contract, those parties that have got the signed contract in the main protocol may need to double check with the TTP about the status of the contract by the deadline t. (Note that the double check does not mean the involvement of the TTP itself, but just a query to a public file maintained by the TTP.) If they do not want to wait until that deadline, they can send the resolve request to the TTP instead, thus blocking other parties to enable the TTP to issue the *cancel* token.

6. CONCLUSIONS

Repudiation is one of the fundamental security threats existing in paper-based and electronic environments. In order to give an answer to such a threat, non-repudiation is defined by the ITU as one of the main security services. As such, this has received relatively enough attention from the scientific community, specially assuring fairness and designing more efficient protocols. But this has been mainly focused towards protocols (or applications) in which two entities take part. Some applications nowadays assume the existence of several participating users and the requirements with respect to fairness and efficiency change significantly. Thus, this service needs to be fully considered in multi-party environments.

We provided a survey which starts from the inherent fundamentals of the non-repudiation service defining the different elements, roles, phases, requirements and state of the art standard analysis of the basic (two-entity oriented) non-repudiation services. This allows us to address the definitions and revisit the requirements of multi-party schemes. In this survey we split multi-party scenarios in two main categories: 1-N and N-N scenarios. With this classification we surveyed the existing work and provided an analysis of the existing research literature.

We stepped forward providing a novel solution for the 1-N multi-party case in which a message can be multicasted to several recipients. We designed our general protocols in a way that allows the participating entities to send personal and confidential messages to other entities without loss of privacy in two different approaches: on-line and optimistic, thus enabling different types of applications. Requirements analysis and comparison in terms of efficiency with previous solutions is provided as well.

We moved towards the application field reading up on applications for multi-party certified electronic mail and contract signing. In both cases we contributed with new solutions. In the first case we generalized a fast and simple optimistic fair certified electronic mail protocol to the multi-party case and integrated and analyzed an asynchronous timeliness property. We also improved existing solutions for synchronous multi-party contract signing protocols based on two differentiated phases: a promise to sign, and a real signature that a party releases only after receiving all promises from the rest of participants. Merging both phases we achieved an improvement on the number of steps with respect to the surveyed literature.

REFERENCES

Asokan, N. 1998. Fairness in electronic commerce. Ph.D. thesis, University of Waterloo, Computer Science.

Asokan, N., Baum-Waidner, B., Schunter, M., and Waidner, M. 1998. Optimistic synchronous multi-party contract signing. Tech. Rep. RZ 3089, IBM Zurich Research Lab.

Asokan, N., Schunter, M., and Waidner, M. 1996. Optimistic protocols for multi-party fair exchange. Tech. Rep. RZ 2892 (# 90840), IBM, Zurich Research Laboratory.

Asokan, N., Schunter, M., and Waidner, M. 1997. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security.* ACM Press, 7–17.

Asokan, N., Shoup, V., and Waidner, M. 2000. Optimistic fair exchange of digital signatures. *IEEE J. Sel. Area. Comm. 18,* 4, 593–610.

Ateniese, G., de Medeiros, B., and Goodrich, M. T. 2001. TRICERT: A distributed certified E-mail scheme. In *Network and Distributed System Security Symposium Conference Proceedings.*

Bao, F., Deng, R., and Mao, W. 1998. Efficient and practical fair exchange protocols with off-line ttp. In *IEEE Symposium on Security and Privacy.* IEEE, 77–85.

Bao, F., Deng, R., Nguyen, K., and Varadharajan, V. 1999. Multi-party fair exchange with an off-line trusted neutral party. In *Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on.* 858–862.

Baum-Waidner, B. 2001. Optimistic asynchronous multi-party contract signing with reduced number of rounds. In *Automata, Languages and Programming*, F. Orejas, P. Spirakis, and J. V. Leeuwen, Eds. 28th International Colloquium, ICALP 2001, Springer-Verlag, 898–911.

Baum-Waidner, B. and Waidner, M. 1998. Optimistic asynchronous multi-party contract signing. Tech. Rep. RZ 3078, IBM Zurich Research Lab.

Baum-Waidner, B. and Waidner, M. 2000. Round-optimal and abuse-free multi-party contract signing. In *27th International Colloquium on Automata, Languages and Programming*. LNCS, vol. 1853. Springer, 524–535.

Blum, M. 1981. Three applications of the oblivious transfer: Part i: Coin flipping by telephone; part ii: How to exchange secrets; part iii: How to send certified electronic mail. Tech. rep., Department of EECS, University of California.

Brannigan, C. 2004. Beyond e-commerce: Expanding the potential of Online Dispute Resolution. *Interaction 16,* 4, 15–17.

Chadha, R., Kremer, S., and Scedrov, A. 2004. Formal analysis of multi-party contract signing. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*. IEEE Computer Society Press, 266–279.

Chiou, G. and Chen, W. 1989. Secure broadcasting using the secure lock. *IEEE Transaction on Software Engineering 15,* 8 (August), 929–934.

DeMillo, R. A. and Merritt, M. 1983. Protocols for data security. *IEEE Computer 16*, 39–50.

Even, S., Goldreich, O., and Lempel, A. 1985. A randomized protocol for signing contracts. In *Communications of the ACM*. Vol. 28. 637–647.

Ferrer-Gomila, J. L., Payeras-Capellà, M., and Huguet-Rotger, L. 2001. Efficient optimistic n-party contract signing protocol. In *Proceedings of the 4th International Conference on Information Security*. Springer-Verlag, 394–407.

Ferrer-Gomila, J. L., Payeras-Capellà, M., and Huguet-Rotger, L. 2002. A realistic protocol for multi-party certified electronic mail. In *Information Security ISC 2002*. LNCS, vol. 2433. 210–219.

Ferrer-Gomila, J. L., Payeras-Capellà, M., and Huguet-Rotger, L. 2004. Optimality in asynchronous contract signing protocols. In *1st International Conference on Trust and Privacy in Digital Business*. Vol. 3184. Springer-Verlag, 200–208.

Franklin, M. and Tsudik, G. 1998. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In *Proceedings of Financial Cryptography 1998*. Lecture Notes in Computer Science, vol. 1465. Springer, 90–102.

Garay, J. A. and MacKenzie, P. D. 1999. Abuse-free multi-party contract signing. In *Proceedings of the 13th International Symposium on Distributed Computing*. Springer-Verlag, 151–165.

González-Deleito, N. 2005. Trust reletionships in exchange protocols. Ph.D. thesis, Faculté des Sciences, Université Libre de Bruselles.

González-Deleito, N. and Markowitch, O. 2001. An optimistic multi-party fair exchange protocol with reduced trust requirements. In *Proceedings of the 4th International Conference on Information Security and Cryptology*. Lecture Notes in Computer Science, vol. 2288. Springer-Verlag, 258–267.

González-Deleito, N. and Markowitch, O. 2002. Exclusion-freeness in multi-party exchange protocols. In *Lecture Notes in Computer Sciences*. 5th International Conference on Information Security (ISC 2002), Springer-Verlag, 200–209.

Gürgens, S. and Rudolph, C. 2002. Security analysis of (un-) fair non-repudiation protocols. In *Formal Aspects of Security*. LNCS, vol. 2629. Spinger-Verlag, 99–114.

Gürgens, S., Rudolph, C., and Vogt, H. 2003. On the security of fair non-repudiation protocols. In *Proceedings of 2003 Information Security Conference*. Vol. 2851. Springer-Verlag, 193–207.

ISO/IEC 1991. *1st WD 13888-2. Non-repudiation using a symmetric key algorithm. JTC1/SC27/WG2 N83*. ISO/IEC.

ISO/IEC 1996. *DIS 10181-4. Information technology - Open systems interconnection - Security frameworks in open systems - Part 4: Non-repudiation*. ISO/IEC.

ISO/IEC 1997. *2nd CD 13888-3. Information technology - Security techniques - Non-repudiation - Part 3: Using asymmetric techniques. JTC1/SC27 N1379*. ISO/IEC.

ISO/IEC 1998. *3rd CD 13888-2. Information technology - Security techniques - Non-repudiation - Part 2: Using symmetric encipherment algorithms. JTC1/SC27 N1276*. ISO/IEC.

ISO/IEC 2004. *13888-1. Information technology - Security techniques - Non-repudiation - Part 1: General model. JTC1/SC27*. ISO/IEC.

ITU-T X.509 2000. *Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks.* ITU-T X.509.

ITU-T X.813 1996. *Information technology - Open Systems Interconnection - Security frameworks for open systems: Non-repudiation framework.* ITU-T X.813.

Khill, I.and Kim, J., Han, I., and Ryou, J. 2001. Multi-party fair exchange protocol using ring architecture model. *Computers & Security 20,* 5, 422–439.

Kremer, S. and Markowitch, O. 2000a. A multi-party non-repudiation protocol. In *Proceedings of SEC 2000: 15th International Conference on Information Security.* IFIP World Computer Congress, 271–280.

Kremer, S. and Markowitch, O. 2000b. Optimistic non-repudiable information exchange. In *J. Biemond, editor, 21st Symp. on Information Theory in the Benelux.* Werkgemeenschap Informatie- en Communicatietheorie, 139–146.

Kremer, S., Markowitch, O., and Zhou, J. 2002. An intensive survey of fair non-repudiation protocols. *Computer Communications 25,* 17 (Nov.), 1606–1621.

Lindell, Y. 2003. *Composition of Secure Multi-Party Protocols.* Springer.

Markowitch, O., Gollmann, D., and Kremer, S. 2002. On fairness in exchange protocols. In *5th International Conference on Information Security and Cryptology*, Springer-Verlag, Ed. LNCS, vol. 2587. 451–464.

Markowitch, O. and Kremer, S. 2000. A multi-party optimistic non-repudiation protocol. In *Proceedings of 3rd International Conference on Information Security and Cryptology.* LNCS, vol. 2015. Springer-Verlag, 109–122.

Markowitch, O. and Roggeman, Y. 1999. Probabilistic non-repudiation without trusted third party. In *Second Workshop on Security in Communication Network 99.*

Markowitch, O. and Saeednia, S. 2001. Optimistic fair-exchange with transparent signature recovery. In *Proceedings of Financial Cryptography 2001.* LNCS, vol. 2339. Springer-Verlag, 339–350.

Maurer, U. 2004. New approaches to digital evidence. In *Proceedings of the IEEE.* Vol. 92. IEEE, 933–947.

Micali, S. 2003. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing.* ACM Press, 12–19.

Mills, D. L. 1992. Network time protocol (version 3) specification, implementation and analysis. Tech. Rep. RFC 1305, IETF Working Group.

Onieva, J. A., Zhou, J., Carbonell, M., and Lopez, J. 2003. A multi-party non-repudiation protocol for exchange of different messages. In *18th IFIP International Information Security Conference. Security and Privacy in the Age of Uncertainty.* IFIP, Kluwer Academic Publishers, 37–48.

Onieva, J. A., Zhou, J., and Lopez, J. 2004. Non-repudiation protocols for multiple entities. *Computer Communications 27,* 16, 1608–1616.

Pfitzmann, B., Schunter, M., and Waidner, M. 1998. Optimal efficiency of optimistic contract signing. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing.* ACM Press, 113–122.

Shao, M.-H., Zhou, J., and Wang, G. 2005. On the security of a certified e-mail scheme with temporal authentication. In Proceedings of 2005 ICCSA Workshop on Internet Communications Security. *Lecture Notes in Computer Science 3482*, 701–710.

Zhou, J. 2001. *Non-repudiation in electronic commerce.* Computer Security Series. Artech House.

Zhou, J. 2004. On the security of a multi-party certified email protocol. In Information and Communications Security: 6th International Conference. *Lecture Notes in Computer Science 3269*, 40–52.

Zhou, J. and Gollmann, D. 1996. A fair non-repudiation protocol. In *Proceedings of IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 55–61.

Zhou, J. and Gollmann, D. 1997. An efficient non-repudiation protocol. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop.* IEEE Computer Society Press, 126–132.

Zhou, J., Onieva, J. A., and Lopez, J. 2005. Optimised multi-party certified email protocols. *Information Management & Computer Security Journal 13,* 5, 350–366.

Zhou, J., Onieva, J. A., and Lopez, J. 2006. A synchronous multi-party contract signing protocol improving lower bound of steps. In *21st IFIP International Information Security Conference Security and Privacy in Dynamic Environments*. IFIP, vol. 201. IFIP SEC, Springer, 221–232.