The Delivery and Evidences Layer

Amir Herzberg and Igal Yoffe

Computer Science Department, Bar Ilan University,
Ramat Gan, 52900, Israel
{herzbea,ioffei}@cs.biu.ac.il

April 4, 2007

Abstract. Evidences of delivery are essential for resolving (and avoiding) disputes on delivery of messages, in classical as well as electronic commerce. We present the first rigorous specifications and provably-secure implementation, for a communication layer providing time-stamped evidences for the message delivery process. This improves on existing standards for evidences ('non-repudiation') services, based on informal specifications and unproven designs.
Our work also improves on the large body of analytical works on tasks related to evidences of delivery, such as certified mail/delivery protocols and fair exchange (of signatures). We improve by addressing practical needs and scenarios, using realistic synchronization and communication assumptions, supporting time-outs and failures, and providing well-defined interface to the higher-layer protocols (application). Furthermore, we use the layered specifications framework, allowing provably-secure use of our protocol, with lower and higher layer protocols, with complete re-use of our analysis (theorems).

Keywords: Certified delivery, cryptographic protocol, fair exchange, layered specifications, non-repudiation, secure e-commerce.

1 Introduction

Reliable, fair and time-stamped message delivery is central to development of e-commerce. Without a doubt, every business relationship between parties includes provisions, whether implicit or explicit, regarding resolution of disputes and the communication mechanism between the trading parties. In addition, when commercial risk is substantial, typically a notarial entity is introduced for the certification and time-stamping of mail.

Currently deployed messaging systems typically do not have a secure, agreed mechanism for fair resolution of disputes regarding communicated messages. Especially in e-commerce scenarios, laws and regulations often force default resolution, e.g. in favor of the buyer, or as an opposite example, in favor of bank records, when client denies a transaction. Though such "thumb-laws" are acceptable in majority of scenarios, admittedly, they bear a potential for unfairness to one of the parties. Moreover, in the majority of the cases dispute resolution involves a human arbiter, which substantially raises the costs of such resolution.

In current work we show how time-stamping and certification service could be provided electronically and in an efficient way. In addition we show how fair dispute resolution is achieved automatically. We believe an *evidence layer* for certified and time-stamped delivery, such as we present, would become an underlying layer for many e-commerce applications and architectures [1, 2].

There has been extensive amount of research on secure certified (electronic) mail, and the related Trusted Third Party (TTP) protocols of fair exchange, contract signing and time-stamping;

we consider all these to be variants of the trusted delivery problem, and refer to the TTP as *notary*. Our protocol extends *Protocol F* of Asokan, Shoup and Waidner [3], and shares many of its properties; in particular it is optimistic - i.e., the notary is involved only if the sender and receiver cannot directly complete the communication due to failure. Our protocol has the following advantages:

We provide precise (and compact) bounds for the termination time and imprecision of time-stamp. Previous works assumed perfect synchrony, or did not provide time-stamp and bounded termination.

We provide the recipient with an affidavit of availability, allowing it to prove that it was operative at give time interval. This is crucial for allocation of liability to failures between the parties. Failures may be due to outages, denial of service attacks, or attempts by the party to ignore unwelcome message.

Our protocol ensures accountability, i.e. if the notary tries to provide services incorrectly, this can be proven based on the affidavits signed by the notary or detected in real time. Note that accountability implies a visible notary, i.e. the identity of the notary can be inferred from the affidavits produced by the protocol. We believe that for many e-commerce applications, e.g. e-banking, accountability is more crucial than invisibility of the notary; however most works do not provide accountability (sometimes, in order to provide invisible TTP [4, 5]). One notable exception is the previously mentioned Protocol F of [3].

Our protocol is simple, practical and efficient, yet proven secure. The efficiency gain in proving protocol security in our layered model is substantial, compared to rigorously proven protocols, e.g. [6].

We address the practical need of parties to get evidences for sending messages in time. This extend the traditional notion of fairness (i.e., with fair exchange sender does not get evidence for reception iff receiver does not get evidence for the message origin).

We specify and analyze the dispute resolution process.

There is considerable amount of literature published regarding fair exchange and non-repudiation, and we would give only a partial and succinct comparison. For comprehensive overviews see [7, 8].

An underlying delivery layer for secure e-commerce applications is part of the architecture of [1], which also addressed the initialization process, mostly focusing on the legal aspects; they considered only fairness goals. There are also several messaging systems and standards offering fair delivery services, in particular [9, 10], and ISO standards [11, 12].

Much of the research on fair delivery focuses on avoiding the use of a notary, by probabilistic or gradual exchange, as in [13, 14]; these works are applicable only to special applications, mainly due to their high overhead and probability of failure; and certainly they do not provide time-stamping services. Additional research direction, is the *optimistic* approach, where the notary is involved only to handle exceptions. There are many works which adopt and study optimistic approach delivery protocols [3–7, 15–17].

The timeliness properties of the protocol imply, in particular, that the 'proofs' produced include a digitally signed time-stamp. The time-stamp allows the receiver (sender) to prove to a third party, e.g. arbiter, that the message was sent (respectively, received) by a particular sender (respectively, receiver) during a specific time interval. The signatures we use in the timestamps are simply those of the sender and receiver, except if the receiver fails, in which case the time-stamp, on the proof that the message was sent and failed to be acknowledged, is by the notary.

Some applications may require additional, non-sender's, signature on a time-stamp to enable long-lasting signatures where exposure of (old) keys could be managed. Our protocol could be used in these cases as more efficient than having the parties contact a Time Stamping Authority, as with the standard Time Stamp Protocol (TSP) [18].

To our knowledge, there are very few works on non-repudiation [19, 8, 20] which embed automatic dispute resolution, or define an appropriate resolution process. The result are designed for humans, and either not fully automated or do not consider the effect of communication failures.

In current work we also handle the confidentiality requirement. We show both variants when confidentiality is preserved by evidence layer, though the notary is not trusted with the content of the messages, and when confidentiality is inherited from lower, transport layer, when notary is trusted.

Notation. Throughout this document, we use **dot notation:** $a.b$, to denote element b of a record or tuple a.

2 Evidence layer

The evidence layer is the lowest secure e-commerce layer. Evidences layer is positioned on top of a transport layer, such as, for example, TCP/IP, TLS/SSL, themselves working on top of socket or SSL API, respectively, and provides additional certification services. An evidence layer session is always between three parties: a client, server, and a notary (trusted third party), which acts as time-stamping and certification provider.

Evidence Field	**Description**
type	Evidences of origin, delivery, failed submission and availability, *EOO, EOD, EOFS, EOA*, respectively.
t	Evidence time-stamp.
msg	The evidence message.
σ	Evidence signature (proof).

Table 1. Evidence structure.

An *evidence* is a time-stamped and signed statement, by an entity or a notary. The evidences are crucial for proving statements regarding the runs of the protocol, for example, that certain messages were delivered to a party. We show an evidence in Table 1. We distinguish four main types of evidences for the evidence layer,

Evidence of origin (EOO), is an attestation of a message send time and origin. The evidence is output by server, which is the recipient of messages, and is signed by the client.

Evidence of delivery (EOD), is an attestation of a message acceptance by an intended message receiver, and the acceptance time. The evidence is output by the client (message sender), and is signed by the server.

Evidence of failure and submission (EOFS), is a complementing evidence for EOD; the evidence states that a message was not acknowledged by the intended receiver. The evidence is output and signed by both client and notary.

Evidence of availability (EOA), is an attestation of server's availability. The evidence is signed by the notary, and includes the time intervals during which the notary did not issue evidences of failure and submission.

Generally, the EOO is intended for the server to prove that the message was sent by the client at specified time. The client, on the other hand obtains an EOD as a proof for the client that the server has indeed received the message. However, when the server does not acknowledge the message the client obtains an EOFS to prove that a message was sent but was not acknowledged, e.g. consider a client placing an stock buy order at a broker. Nevertheless, server may prove it did acknowledge all messages at some time interval by obtaining an EOA from the notary for that interval. Of course, having both EOFS and EOA with overlapping times means that either the notary was compromised, e.g. notary's signature was forged.

Time-stamping. The evidence layer supplies, as part of the aforementioned evidences mentioned above, attestation of message existence at specific time interval, and attestation that by that time it was also signed by the originating party. This allows non-repudiation of a message even after message originator's key had expired, been compromised or revoked.

Confidentiality. In current work we treat confidentiality twofold. First, the evidence layer may provide confidentiality (in the indistinguishability sense) by itself. On other hand confidentiality could be solved by cautious use of a confidentiality providing transport or communication layers, for instance, TLS/SSL or IP-Sec layers. In later sections we show how to define confidentiality specifications for both cases.

Failed Delivery. We assume simple management of non-delivered messages, where each message is assigned only one type of evidence during lifetime. The evidence layer does not try to re-deliver messages (reliability service could be provided by layers below the layer of evidences, e.g., TCP), if it had been already assigned an EOFS.

2.1 Agreements, interfaces and validation

We proceed to define the evidence layer, by first defining the concept of an agreement, which provides the context for the communication and evidences. We then proceed to define the precise interfaces exposed by the evidence layer, and finally, we discuss the validation mechanism to resolve disputes between the communicating parties.

Agreements. Any communication between parties always involves a context and an agreement, whether implicit or explicit. An *agreement*, shown in Table 2, is an accord between an attestation client, notary and server, i.e. evidence layer channel participants, regarding the communicating parties addresses, parties validation and encryption keys, clock drifts and bounds, and lower layer communication information, such as the allowed messages and maximum allowed message delay.

Remark 1. We have introduced the communication validity function, in Table 2 to capture practical considerations of communication protocols working with limited or predefined sets of messages. A trivial example would be a function that limits the length of the sent messages, i.e., $V_{comm}(m) = (|m| \leq c)$, for some $c \in \mathbb{N}$.

Agreement Fields	Description
C.addr, C.vk, C.ek, S.addr, S.vk, S.ek, N.addr, N.vk, N.ek	The identities of the principals participating in the agreement; client, server and notary, respectively. Principal's identity is an *(addr,vk,ek)* tuple, of principal's address and public signature validation and encryption keys.
α,β	Clocks drifts and biases bounds, respectively.
Δ_{comm}	Communication delay bound, between any two parties.
V_{comm}: $\{0,1\}^* \mapsto \{\texttt{true}, \texttt{false}\}$	Communication message validity test function.

Table 2. Evidences layer agreement.

Interfaces. We show precise interfaces for the evidence layer in Tables 3–7. In Table 3 we show the interfaces between the evidence layer and the user, i.e. application. In Table 4 we show the interface with the communication service layer, and in in Tables 5 and 6 respectively, we show the interface with the signature and encryption schemes used by the layer. Additional interface with a clock service, is shown in Table 7.

Method	Direction	Description
EL.Init(1^k,*addr*,α,β,Δ_{comm},V_{comm}) *EL.InitResult*(*vk*,*ek*,Δ_{el})	in out	Initializes layer, with security parameter, 1^k, and lower layer settings, see Table 2. Returns `false` if initialization failed, or returns the generated validation key *vk*, public encryption key *ek*, and layer's bound Δ_{el} on delivery of evidences (Table 1).

EL.OpenChannel(Agr,ρ)
 	
EL.OpenChannelResult(Agr,status)
	in
	
	out
		Establishes an attested channel for the role $\rho \in \{S, C, N\}$, client, server and notary, respectively.
		Return a boolean *status* channel open result.

EL.Send(Agr,m)
EL.SendResult(Agr,e,h)
	in
	out
		Sends a message *m* on an open channel identified by *Agr* agreement.
		Returns an EOD or EOFS (Table 1), for previously sent message on channel identified by *Agr* agreement, or a special *FAIL* indication of channel failure. The field *h* is a hint, for encrypted evidence content.

EL.AvailabilityRequest(Agr)	in	Request EOA from the notary.
EL.Receive(Agr,e,h)	out	Delivers evidence *e*, which could be an EOA, or EOO (Table 1). Also used for channel failure, $e = FAIL$ notification. The field *h* is a hint, for encrypted evidence content.

Table 3. Evidences layer interface with application.

Remark 2. We note that the specified interfaces, e.g. the encryption interface, lack the explicit option for randomness, since the randomness is implicitly supplied for each protocol event by the execution framework [21, 22].

Validation. A central element of the evidence layer is the mechanism for resolving disputes between the parties, or in other words, how to validate claims of the parties related to the runs of the protocol. The sender may want to claim (or prove) that she submitted the message on time; that the message was delivered to the destination, who made a specific response, on given

Method	Direction	Description
$Comm.Init(addr,\Delta_{comm},V_{comm})$	in	Initializes a communication channel using *addr* address, delivery delay bound Δ_{comm} and V_{comm}message validity function.
$Comm.InitResult(status)$	out	Outputs whether the communication service was successfully initialized.
$Comm.Send(addr,m)$	out	Sends a message m, to a party identified by *addr* address.
$Comm.Receive(addr,m)$	in	Delivery of a message m which had arrived from party identified by *addr* address, in addition m could be a special *FAIL*, channel failure indication value.

Table 4. Evidences layer interface with communication layer.

Method	Direction	Description
$\mathcal{S}.Gen(1^k)$	in	Key generation takes an unary security parameter k and a random string $r \in \{0,1\}^k$.
$\mathcal{S}.GenResult(sk,vk)$	out	Outputs the (private) signing key identifier *sk* and a (public) verification key *vk*.
$\mathcal{S}.Sign(sk,m)$	in	Signs a message m using signing key identified by *sk*.
$\mathcal{S}.SignResult(\sigma_m)$	out	Returns the resulting signature σ_m.
$\mathcal{S}.Verify(vk,m,\sigma_m)$	in	Verifies using *vk* key, if message m matches the signature σ_m.
$\mathcal{S}.VerifyResult(b)$	out	Returns a boolean verification result $b \in \{\texttt{true}, \texttt{false}\}$.

Table 5. Evidences layer interface with signature scheme $\mathcal{S}$.

time; or that the destination failed to receive the message. The recipient may want to claim (or prove) that it received the message from the sender (at given time); or that it did not fail to receive any message during given time interval. To facilitate validation of claims, the parties sign relevant evidences during the run. We require the evidence layer to include an efficient $\texttt{Validate}(Agr, e, m, h)$ predicate, to decide whether the evidences are valid or not. The arguments to the validation predicate include an agreement Agr, an evidence e, the message m in the case the message does not equal to $e.msg$, and a hint h, e.g. the randomness that was used to create an encrypted $e.msg$ part. The validation predicate is then to check if the evidence is valid in the context of the Agr agreement, with respect to the input message and hint. Later on, in Section 2.4 we show a specific implementation of the validation predicate, with respect to our implementation.

2.2 Simple evidence layer protocol

In the current section we present an informal description of the *simple evidence layer protocol* (SELP). We do present the protocol implementation code in Appendix. The protocol is natural and straightforward for either physical security (envelops and seals) or cryptographic security. In later section we present an optimistic variant, *optimistic evidence layer protocol (OELP)*, as simple supplement to SELP.

SELP participants are client, notary and server, and all are involved in the protocol execution. There are four basic types of inner flow types in SELP. The '*M*' (for 'Message') is the flow which depicts the message sent and signed by the client and forwarded by the notary to the server; the '*R*' (for 'Response') flow depicts the server's signed response, forwarded by the notary to the client; the '*RA*' (for 'Response acknowledgment') flow is the signed acknowledgment by the notary to the server, regarding server's response. Additional flow is the '*F*' (for 'Failure') flow where the notary affirms the client that no response was received from the server.

Method	**Direction**	**Description**
$\mathcal{CE}$.Gen(1^k)	in	Key generation takes an unary security parameter k and a random string $r \in \{0,1\}^k$.
$\mathcal{CE}$.GenResult(dk,ek)	out	Outputs the (private) decryption key identifier *dk* and a (public) encryption key *ek*.
$\mathcal{CE}$.Enc(ek,m)	in	Encrypts a plaintext m using encryption key *ek*.
$\mathcal{CE}$.EncResult(c)	out	Returns the resulting ciphertext c.
$\mathcal{CE}$.Dec(dk,c)	in	Decrypts a ciphertext c using decryption key identified by *dk*.
$\mathcal{CE}$.DecResult(m,r)	out	Returns the corresponding plaintext m and the randomness r that was used in the encryption process.

Table 6. Evidences layer interface with encryption scheme $\mathcal{CE}$.

Method	**Direction**	**Description**
Clock.Init(α,β)	in	Initializes the clock services, with global drift bound α and bias bound β.
Clock.InitResult(status)	out	Outputs whether the communication service was successfully initialized.
Clock.SetTimer(cky,offset)	out	Schedules a wakeup call, identified by unique cookie *cky*, in relative *offset* time from the party's current clock value.
Clock.CancelTimer(cky)	out	Cancels a schedules a wakeup call, identified by unique cookie *cky*.
Clock.WakeAlarm(cky)	in	Notification of a scheduled wakeup, identified by *cky* cookie. The wakeup event cancels the timer.

Table 7. Evidences layer interface to clock.

Faultless scenario. We show the basic, sunny-day faultless flows in Figure 1. The figure guidelines the protocol implementation in Appendix, assuming that each pairwise communication takes place within the Δ_{comm} underlying communication channel delivery bound. The process is initiated by the client at time t_{C_1}, sending message m to the server. The client signs the (*'M'*,t_{C_1},*m*) tuple as σ_M. When the message arrives at time t_{N_1} to the notary, the notary validates client's signature and forwards the message as is to the server (recording the forward time). After the message arrives to the server, at time t_{S_1} at the figure, the server signs an (*'R'*,t_{C_1},*m*) reply as σ_R, and replies with this signed tuple to the notary. The notary, receiving the reply at t_{N_2}, forwards it to the client after validating server's signature and acknowledges the server by replacing server's signature with its own σ_{RA}. The client receives the reply at t_{C_2}, validates server's signature and terminates. We note that in the sunny-day scenario just described, all signatures are valid and replies are received within expected bounded time, e.g. $|clk_N(t_{N_1}) - clk_C(t_{C_1})| \leq \Delta_{comm}$, where $clk_p(t)$ denote the real-time clock reading t of party p (which clock may drift, and be initially biased). When all signature are valid and all messages received in time, as in the currently discussed scenario, and evidence of origin is produced by the server, and an evidence of delivery is produced by the client. The server assembles the EOO evidence, with σ_M and σ_{RA} as the proof that, respectively, the message did originate from the client at specified t_{C_1} time, and that he provided his part of the fair deal. The client assembles an EOD evidence with σ_M and σ_R signature, as the proof that message is his own and was sent at the specified time, and that this exact information was also acknowledged by the server.

No acknowledgment from server. In current scenario, depicted in Figure 2, the notary does not receive a reply from the server within $2 \cdot \Delta_{comm}$, and issues an EOFS evidence for the client. To do so the notary signs an (*'F'*,$t^F_{C_1}$,*m*) tuple as σ_F, where $t^F_{C_1}$ is the time the client had sent the message, and send the tuple with the signature back to the client. The EOFS proof

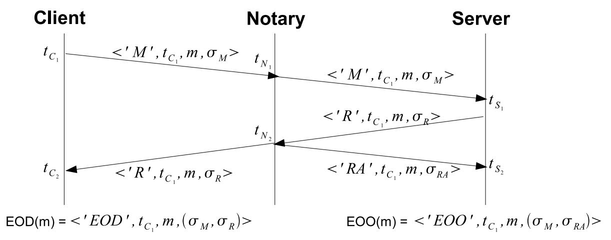

Client
Notary
Server
t_{C_1}
$<'M', t_{C_1}, m, \sigma_M>$
t_{N_1}
$<'M', t_{C_1}, m, \sigma_M>$
t_{S_1}
$<'R', t_{C_1}, m, \sigma_R>$
t_{N_2}
t_{C_2}
$<'R', t_{C_1}, m, \sigma_R>$
$<'RA', t_{C_1}, m, \sigma_{RA}>$
t_{S_2}
EOD(m) = $<'EOD', t_{C_1}, m, (\sigma_M, \sigma_R)>$
EOO(m) = $<'EOO', t_{C_1}, m, (\sigma_M, \sigma_{RA})>$

Fig. 1. SELP flow without faults. Client sends a valid message, and obtains an *evidence of delivery*, formed from the *'M'* and *'R'* messages signatures. In similar way the server obtains *evidence of origin* formed from the *'M'* and *'RA'* messages signatures.

constitutes then from both the client's signature on the *'M'* message and notary's signature on the *'F'* message.

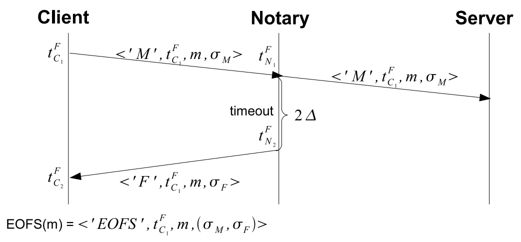

Client
Notary
Server
$t^F_{C_1}$
$<'M', t^F_{C_1}, m, \sigma_M>$
$t^F_{N_1}$
$<'M', t^F_{C_1}, m, \sigma_M>$
timeout
2Δ
$t^F_{N_2}$
$t^F_{C_2}$
$<'F', t^F_{C_1}, m, \sigma_F>$
EOFS(m) = $<'EOFS', t^F_{C_1}, m, (\sigma_M, \sigma_F)>$

Fig. 2. SELP flow with server failure. Client sends a valid message, and obtains an *evidence of failure and submission*, formed from the *'M'* and *'F'* messages signatures. The server does not obtain any evidence.

Server's availability. Last scenario type is the 'A' (for 'Availability') flow. As previously mentioned, the server might want to show that it was available during certain time intervals, thus effectively disputing notary's honesty in the case an EOFS is presented by the client for the interval server was available. We show the availability flow in Figure 3. The notary responds to server's availability request with a signed (*'A'*,$(t_1, t_2]$) tuple. The times t_1 and t_2 represent the interval which is approved by the notary as EOFS free (in the context of the evidence layer agreement).

Though, in Figures 1–3 flows we did not demonstrate what happens when the notary fails, the implementation code in Appendix handles this case. In fact, the notary may fail or exhibit adversarial behavior. In particular the notary may fail to return an acknowledgment or to forward a response. The client detects that it did not receive an 'R' or an 'F' message within reasonable time. Similarly, the server detects that it did not get an 'RA' response message from the notary. However, such faults of the notary could not be proven to an arbiter and the only recourse for the parties is to raise a 'FAIL' alert for upper (application) layer, as soon as failure of the notary is detected.

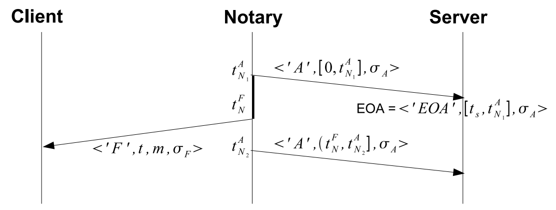

Client
Notary
Server
$t^A_{N_1}$
$<'A',[0,t^A_{N_1}],\sigma_A>$
t^F_N
EOA = $<'EOA',[t_s,t^A_{N_1}],\sigma_A>$
$<'F',t,m,\sigma_F>$
$t^A_{N_2}$
$<'A',(t^F_N,t^A_{N_2}],\sigma_A>$

Fig. 3. SELP flow for availability. Server obtains an *evidence of availability*, formed directly from the *'A'* message. Evidences of availability are not issued for time interval between previous such evidence and *'F'* failure notification message, to avoid claims of notary dishonesty. Boldface line part indicates the time interval for which no availability evidence is issued by the notary.

2.3 Optimistic evidence layer protocol

In OELP the notary is involved only in the failure scenario, at which point the protocol falls back to SELP. Consider a faultless scenario for sending a message m, described in Figure 4. The difference from Figure 1 is that the client is the one signing the 'RA' messages. When the client does not receive an 'R' response for an 'M' protocol message it had sent, the game is fair, as nor client nor server could create a valid evidence, each missing a σ_M and σ_R signatures on the message m, respectively. However, the server is in an inferior position, when not candid client does not send an 'RA' message, as the client could create an EOO evidence for m, while the server could not yet create an EOD for it. This situation is described in Figure 5, and server recovery is a straightforward fallback to SELP by involving the notary and resending it the 'R' message. When the notary replies with an 'RA' answer, the server could also create an EOD evidence.

Remark 3. We note that many fair exchange and non-repudiation protocols, e.g. [4, 23–25] are concerned with obtaining the same proof (signature) for messages whether the notary was involved or not (this is also known as *transparent* signatures). We do not view this mission as central to our goal. In our opinion what is central in e-commerce is the validity of the evidence itself, e.g. valid evidence of failure and submission for cheque deposit in a bank could be later submitted for additional credit. In the arbitration process initiated by layers above evidence layer, the `Validate` function would be used to conduct evidence validation for SELP or OELP.

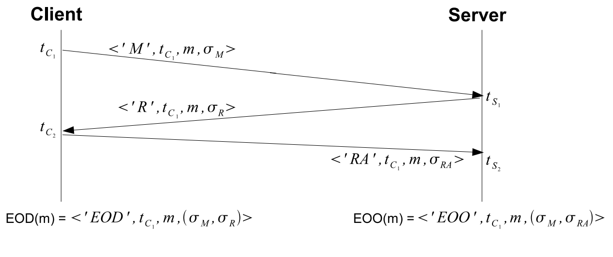

Client
Server
t_{C_1}
$<'M',t_{C_1},m,\sigma_M>$
t_{S_1}
$<'R',t_{C_1},m,\sigma_R>$
t_{C_2}
$<'RA',t_{C_1},m,\sigma_{RA}>$
t_{S_2}
EOD(m) = $<'EOD',t_{C_1},m,(\sigma_M,\sigma_R)>$
EOO(m) = $<'EOO',t_{C_1},m,(\sigma_M,\sigma_{RA})>$

Fig. 4. OELP flow without faults. Notary is not involved, and client and server parties obtain the relevant evidences, *evidence of delivery* and *evidence of origin* respectively.

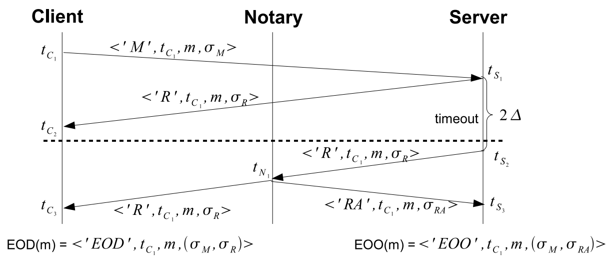

Client
Notary
Server
t_{C_1}
$<'M', t_{C_1}, m, \sigma_M>$
t_{S_1}
$<'R', t_{C_1}, m, \sigma_R>$
timeout
2Δ
t_{C_2}
$<'R', t_{C_1}, m, \sigma_R>$
t_{S_2}
t_{N_1}
t_{C_3}
$<'R', t_{C_1}, m, \sigma_R>$
$<'RA', t_{C_1}, m, \sigma_{RA}>$
t_{S_3}
EOD(m) = $<'EOD', t_{C_1}, m, (\sigma_M, \sigma_R)>$
EOO(m) = $<'EOO', t_{C_1}, m, (\sigma_M, \sigma_{RA})>$

Fig. 5. OELP flow with faults and fall-back to SELP by server, indicated by dotted line. While client possibly have obtained an *'R'* message and could complete an *evidence of delivery*, server did not obtain an *'RA'* message and to complete an *evidence of origin* contacts the notary. The notary sends both *'RA'* and *'R'* messages, to prevent fake or premature server requests for *'RA'*.

2.4 Validation of evidences

Admittedly, a clear and automated process by which parties' evidences are validated is crucial. In Figure 12 we show an implementation for validation function `Validate` for SELP. The arguments to the validation are the agreement, validated evidence, and in the case encryption is used a message and hint (e.g. randomness used during the encryption process). In all evidence types the validation is similar: verifying that the encrypted content of the evidence corresponds to the supplied message, and validating signature of the signer of the corresponding evidence part (client's signature for 'M' message, server's signature for 'R' message, and notary's signature for failures 'F' and response acknowledgments 'RA' messages). For brevity, we omit implementation for OELP, which is similar.

3 Evidence layer specifications

In the current section we informally present the specifications we are to consider for the evidence layer. We adopt a layered analysis [21], which requires that we present the specifications on the interfaces of the analyzed layer, to later show that if lower layer specifications (defined in a similar way) are upheld, then so are the specifications of the evidence layer. The full model details are available at [21].

Though our model is formal, we have chosen first to informally exhibit the specifications, to capture the essence of the requirements from an evidence layer. Therefore we begin with the more interesting, by our own admission, correctness specifications, proceed with liveness specifications, discuss two confidentiality (indistinguishability) modes and omit the less interesting initialization specifications.

3.1 Correctness specifications

We now define adversarial win predicates on the evidence layer interfaces. The predicates would define whether the evidence layer interfaces exhibited **incorrect** behavior with respect to protocol execution.

In forging and evidence of origin for a message, we consider the following improper on the interfaces of the application layer. The protocol execution consists of a honest client, honest notary and adversarial server. The settings are described in Figure 7(a). We say that an EOO was forged for a message m, if at the end of the protocol execution the adversary could output an evidence layer agreement, a message and an EOO evidence for the message m, denoted *EOO(m)* in the figure (and similar notation follows in subsequent figures), such that the EOO is successfully validated with respect to the evidence layer agreement, however m was not sent by the client over the evidence layer channel associated with the agreement.

We now provide some intuition how to formally describe this condition with a predicate over an execution of a protocol with interfaces as defined in Section 2. First, recall that execution of composite protocol is a view of events produced by both adversarial (corrupted) and non-adversarial parties. The predicate would contain explicit reference to a *Receive* event output by the corrupted server, which contains a valid EOO evidence e, i.e., evidence which passes `Validate` check. The predicate would be upheld if for the former evidence, there was no *Send* event with the message m described by the evidence, at the time described by evidence's time-stamp.

Definition 1 (Forging evidence of origin, `FakeEOO`, informally). *Predicate* `FakeEOO` *is* `true` *for evidence layer protocol execution, if an adversary outputs an evidence layer agreement and a valid EOO evidence, such that the message described by the evidence was not sent by the client, at the time described by the evidence.*

Notice that we have omitted from `FakeEOO` the obvious preconditions that the client and notary parties, as specified by the agreement, have opened an evidence layer channel defined by the agreement, in their (correct) respective roles. We will reintroduce these and other preconditions later, in Section 4.

Similarly, successfully forging an evidence of delivery involves an corrupted client, and a server such that an EOD evidence which passes `Validate` check exists, however, the server have never obtained an EOO evidence for the message. We illustrate this situation in Figure 7(b).

Definition 2 (Forging evidence of delivery, `FakeEOD`, informally). *Predicate* `FakeEOD` *is* `true` *for evidence layer protocol execution, if an adversary outputs an evidence layer agreement and valid EOD evidence, such that there was no receive event, for the message described by the evidence, at the time described by the evidence, with a valid EOO evidence for the message, at the server.*

Forging an EOFS involves a corrupted server. In such case, as described in Figure 7(c), the server would output a valid EOFS evidence, while such was not output by a honest notary.

Definition 3 (Forging evidence of failure and submission `FakeEOFS`, informally). *Predicate* `FakeEOFS` *is* `true` *for evidence layer protocol execution, if an adversary outputs an a valid EOFS evidence such that the notary did not output the same evidence.*

Additional basic correctness specifications is a basic requirement not to provide evidence on the evidence layer API which fail the `Validate` validation. Thus the layer should not provide unchecked content to upper layer.

Definition 4 (Invalid receive and send result, `InvalidReceive` and `InvalidSend`, informally). *Predicate* `InvalidReceive` *(*`InvalidSend`*) is* `true` *for evidence layer protocol execution, if a party has supplied an evidence for application, as part of receive (send result) event, such that the evidence fails the evidence layer validation function.*

We omit the correctness specifications for evidences of EOA type, which are quite similar to the correctness specifications of EOFS evidences. We also remark that we have always considered a honest notary, to avoid forced optimistic implementation for evidence layer protocols.

3.2 Liveness specifications

Liveness specification for the evidence layer are about getting evidences if different parties are connected. Admittedly, if client and notary are connected (and both are non-corrupted) then the client should always receive at least an EOFS evidence, signed by the notary, for sent messages. We describe this requirement in Figure 13(a). Informally,

Definition 5 (Client gets evidence, `LinkClientNotary`, informally). *Predicate* `LinkClientNotary` *is* `true` *for evidence layer protocol execution, if client's invocation of* Send *interface event results in client's* SendResult *event, which contains a valid evidence, within Δ_{el} (bound returned by* InitResult*) time.*

When the server and notary is connected, the situation is not symmetrical, as not the client and server roles. The (non-corrupted) server is ought to expect that a non-failing channel to a (non-corrupted) notary guarantees that no EOFS type evidences would be affirmed by the latter. We describe this requirement in Figure 13(b). Informally,

Definition 6 (No EOFS type evidences, `LinkServerNotary`, informally). *Predicate* `LinkServerNotary` *is* `true` *for evidence layer protocol execution, if no* SendResult *interface event is produced by the notary, with valid EOFS evidence, during the time the channel between server and notary did not fail.*

However, when all the parties are connected, we could expect more than mere combination of the previous requirements. When all parties are connected we could expect that sent message are actually delivered, whether via notary, or alternatively, in an optimistic implementation, via the direct link between the client and the server. Since sent messages are delivered, the server is ought to get an EOO for a delivered message, and the client is ought to get an EOD for it. We picture this requirement in Figure 13(c). Informally,

Definition 7 (Sent messages delivered, `Link`, informally). *Predicate* `Link` *is* `true` *for evidence layer protocol execution, if client's* Send *interface event, results in two* SendResult *events, both within Δ_{el} time interval. A client's* SendResult *interface event with a valid EOD evidence for the message, and server's* SendResult *event with a valid EOO evidence for the message.*

3.3 Confidentiality specifications

Typical engineering practice of protocol design is to design protocols using lower layer protocols as building blocks. In such modular design security and correctness properties of the new protocol are typically derived from the fact that the auxiliary protocols (or their ideal counterparts) uphold some specifications by themselves. Thus deriving protocol specifications for modular design is a bottom-up process where protocols are shown to uphold some specifications as long as protocols used as building blocks uphold some, possibly different, specifications.

In [21] we have provided a formalization for the above process including a composition theorem for specifications. In particular, a test protocol is a particular case of a specification which tests another protocol. In such settings, in addition to the traditional adversary, a test environment may also include components (which implement some other, non-tested protocols) that are controlled by adversary but uphold some specifications. So generally, when testing protocols, we identify four types of participants, viz., (1) processors of the tested protocol, (2) processors of the tester protocol, (3) adversary, and (4) adversarial processors, running non-tested protocol, which nevertheless uphold certain specifications.

Next in this section, we provide the confidentiality (indistinguishability) specifications, as test settings, for two cases,

Untrusted notary. When notary is not trusted, informally, it means that the only two honest players, the evidence layer client and server, are to pass a confidentiality (indistinguishability) game, when connected to an adversary which also acts as a message transport. We note that practically, it means that to pass such confidentiality settings the evidence layer protocol would have to encrypt the application messages.

Trusted notary. When notary is trusted, it is not corrupted by the adversary, and the notary could view the content of the messages. Thus, confidentiality is 'inherited' from the transport below evidence layer. Informally, evidence layer preserves confidentiality, if is placed on top of a confidentiality preserving transport. We note that practically, the meaning of trusted notary, is that the evidence layer protocol itself does not encrypt the content of the application messages.

A practical implementation of a communication protocol (e.g., IPsec) would typically employ encrypt-then-authenticate paradigm, e.g. see [26] why only encryption is not enough. However, it is widely known that extending encryption with additional cryptographic modules may result in `IND-CCA2` test failure, e.g. see [27]. Therefore we use the adaptive replayable chosen ciphertext attack (rCCA) [28] weaker variant of the indistinguishability test for our protocol.

We now informally define the `rCCA` *test configuration*, $C^{\text{NOTRUST}} = \langle \Pi, \mathcal{W}, \Gamma \rangle$, for $\mathcal{ELP}$, evidence layer protocol, when notary is untrusted (corrupted). Let $\mathbb{P} = \{p_{el}^{\text{client}}, p_{el}^{\text{server}}, p_{el}^{\text{test}}\}$ denote a set of participating processors, where p_{el}^{test} mimics all the evidence layer interfaces and has an additional test-result interface. Let $\mathcal{W}$ be their wiring as shown in Figure 15(a) and Figure 14. Additionally, let $\Gamma = \{(p_{el}^{\text{test}}, \mathcal{T}_{\texttt{rCCA}}^{\mathcal{ELP}:C,S}), (p_{el}^{\text{client}}, \mathcal{ELP}), (p_{el}^{\text{server}}, \mathcal{ELP})\}$ be the implementation in the configuration. In following definition we describe the test settings,

Definition 8 ($\mathcal{T}_{\texttt{rCCA}}^{\mathcal{ELP}:C,S}$ test protocol machine in $C^{\textbf{NOTRUST}}$). *A $\mathcal{T}_{\texttt{rCCA}}^{\mathcal{ELP}:C,S}$ IND-rCCA test protocol for evidence layer protocol $\mathcal{ELP}$ is a state machine, running with adversary $\mathcal{A}$ as follows:*

1. *Initialize two evidence layer service instances p_{el}^{client} and p_{el}^{server} and pass the respective initialization return values to $\mathcal{A}$. Receive agreement* Agr *from $\mathcal{A}$ and open evidence layer channels for p_{el}^{client} as a client and p_{el}^{server} as a server, respectively.*
2. *Repeat:*
   (a) If $\mathcal{A}$ outputs $\langle$Send,Agr,m$\rangle$ *invoke* p_{el}^{client}.Send(Agr,m).
   (b) If $\mathcal{A}$ outputs $\langle$Choose,Agr,m$_0$,m$_1\rangle$ *invoke* p_{el}^{client}.Send(Agr,m$_b$) *where $b \in_R \{0, 1\}$.*
   (c) If p_{el}^{client} outputs SendResult(Agr,e,h) *and (*$\texttt{Validate}(Agr, e, m_0, h) = \texttt{true}$ *or* $\texttt{Validate}(Agr, e, m_1, h) = \texttt{true}$*) output REPLAY to $\mathcal{A}$, otherwise pass the output to $\mathcal{A}$ as is.*
   (d) If p_{el}^{server} outputs Receive(Agr,e,h) *and (*$\texttt{Validate}(Agr, e, m_0, h) = \texttt{true}$ *or* $\texttt{Validate}(Agr, e, m_1, h) = \texttt{true}$*) output REPLAY to $\mathcal{A}$, otherwise pass the output to $\mathcal{A}$ as is.*

   Until $\mathcal{A}$ outputs $\langle$Guess,$\hat{b}\rangle$ *where $\hat{b} \in \{0, 1\}$. Return the boolean value of $b = \hat{b}$ on the test-result interface.*

We now informally define the `rCCA` test configuration, $C^{\text{TRUST}} = \langle \Pi, \mathcal{W}, \Gamma \rangle$, for $\mathcal{ELP}$, evidence layer protocol, when notary is trusted (corrupted). Let $\mathbb{P} = \{p_{el}^{\text{client}}, p_{el}^{\text{server}}, p_{el}^{\text{notary}}, p_{el}^{\text{test}}, p_{comm}^{\text{client}}, p_{comm}^{\text{server}},$

$p_{comm}^{\text{notary}}\}$ denote a set of participating processors, where p_{el}^{test} mimics all the evidence layer interfaces and has an additional test-result interface, and the communication (adversarial) processors implement a basic communication interface, with ports for initialization and send and receive, as discussed in [21].

Let $\mathcal{W}$ be their wiring as shown in Figure 15(b). Let Γ be defined similarly to above and augmented with the communication processors implementing some adversarial communication protocol $\mathcal{CP}$. In following definition we describe the test settings,

Definition 9 ($\mathcal{T}_{\mathtt{rCCA}}^{\mathcal{ELP}:C,S,N}$ test protocol machine for $C^{\mathbf{TRUST}}$). *A $\mathcal{T}_{\mathtt{rCCA}}^{\mathcal{ELP}:C,S,N}$ IND-rCCA test protocol for evidence layer protocol $\mathcal{ELP}$ is a state machine, running with adversary $\mathcal{A}$ as follows:*

1. *Initialize three evidence layer service instances p_{el}^{client}, p_{el}^{server} and p_{el}^{notary} and pass the respective initialization return values to $\mathcal{A}$. Receive agreement* Agr *from $\mathcal{A}$ and open evidence layer channels for p_{el}^{client} as a client, p_{el}^{server} as a server, and p_{el}^{notary} as a notary, respectively.*
2. *Continue as in Definition 8-2, with the addition that if p_{el}^{notary} outputs* SendResult(Agr,e,h) *and ($\mathtt{Validate}(Agr, e, m_0, h) = \mathtt{true}$ or $\mathtt{Validate}(Agr, e, m_1, h) = \mathtt{true}$) output $REPLAY$ to $\mathcal{A}$, otherwise pass the output to $\mathcal{A}$ as is.*

Remark 4. We note that intentionally there is no explicit initialization of the communication machines in the above Definition 9, since the test machine is not connected to the communication machines.

Using composition of specifications, we would later claim that if $\mathcal{CP}$ preserves pairwise confidentiality - evidence layer confidentiality, as defined by the test, is also preserved.

4 Analysis

We analyze the two evidence layer protocols, SELP and OELP, we have defined in previous Sections 2.2,2.3, with respect to the generic specifications on the interfaces of evidence layer protocols, defined in Section 2.1. In the following theorems we would use predicates for lower layer which we have defined elsewhere [21]. The predicate `CommGuaranteedDelivery` defines guaranteed delivery on the interfaces of the lower layer, i.e. if a message was sent it would be delivered (but not necessarily vice versa) within bounds provided during initialization of the (lower) communication layer. The predicate `ClockSync` describes synchronized clocks of the non-corrupted parties, with bounds for drifts and initial offset. Finally we make two additional notes, first we note that for brevity we omit discussion of initialization, second, for simplicity, we do not consider cases when the notary is corrupted. For example the specification regarding the exchange of EOO and EOD evidence (Definition 7) could be amended to include the case where only the client and server parties are honest and maintaining a direct communication link. However, this would *force* optimistic implementation for every protocol which is to uphold our specifications.

We now proceed to state a number of theorems regarding our protocol(s).

Theorem 1 (Liveness). *Let $X \in \mathbb{X}[\mathcal{ELP}]$ be an execution of the evidence layer protocol $\mathcal{ELP} \in \{SELP, OELP\}$. Then,*

$$\begin{aligned}&\mathtt{CommGuaranteedDelivery}(X) \wedge \mathtt{ClockSync}(X) \Rightarrow\\ &\quad \mathtt{LinkClientNotary}(X) \wedge \mathtt{LinkServerNotary}(X) \wedge \mathtt{Link}(X)\end{aligned}$$

Theorem 2 (Correctness). *Let evidence layer protocol $\mathcal{ELP} \in \{SELP, OELP\}$ be implemented with CCA-secure encryption scheme $\mathcal{CE}$, CMA-secure signature scheme $\mathcal{S}$ and clock scheme which securely upholds* `ClockSync`*. Then $\mathcal{ELP}$ securely prevents*

1. `FakeEOO`*, if client and notary are not corrupted.*
2. `FakeEOD`*, if server and notary are not corrupted.*
3. `FakeEOFS`*, if notary is not corrupted.*
4. `InvalidReceive` *(*`InvalidSend`*) for non-corrupted party.*

Theorem 3 (Confidentiality with untrusted notary). *Let evidence layer protocol $\mathcal{ELP} \in \{SELP, OELP\}$ be implemented with rCCA-secure encryption scheme $\mathcal{CE}$ and CMA-secure signature scheme $\mathcal{S}$. Then $\mathcal{ELP}$ securely upholds $\mathcal{T}_{\texttt{rCCA}}^{\mathcal{ELP}:C,S}$.*

The following theorem states that when the lower communication layer preserves confidentiality, tested with $\mathcal{T}_{\texttt{rCCA}}^{\mathcal{CP}}$ in [21], the evidences layer also preserves confidentiality, as described by the $\mathcal{T}_{\texttt{rCCA}}^{\mathcal{ELP}:C,S,N}$ configuration.

Theorem 4 (Confidentiality with trusted notary). *Let evidence layer protocol $\mathcal{ELP} \in \{SELP, OELP\}$ be implemented over a communication protocol $\mathcal{CP}$, such that $\mathcal{CP}$ securely upholds $\mathcal{T}_{\texttt{rCCA}}^{\mathcal{CP}}$. Then $\mathcal{ELP}$ securely upholds $\mathcal{T}_{\texttt{rCCA}}^{\mathcal{ELP}:C,S,N}$.*

For shortness of this submission we omit the direct but somewhat tedious proofs of the theorems. We note that the proofs for Theorems 1-3 are derived in a direct manner from examinations of the protocol implementation code, provided in the Appendix, assuming lower layer services (communication and clock) uphold the predicates of clock synchrony and guaranteed (bounded) delivery. However, unlike straightforward (single-layer) proofs, in multi-layer settings of Theorem 4 we additionally apply a secure composition theorem from [21] to show that when the lower communication layer upholds the confidentiality test (when certain specifications are upheld by layer lower than communication) the evidence layer also upholds its confidentiality test.

5 Conclusions

We presented specifications, and practical and provably secure implementations, for the 'delivery with evidences' layer. Delivery with evidences is a basic service required in many e-commerce scenarios, e.g. business-to-business transactions. Our protocol is the first to provide evidences with timestamps and bounded termination under realistic adversarial assumptions. The protocol is used as a foundation layer in an architecture for secure e-commerce, we have described in [2], and as an underlying layer for secure orders layer [22]. Our specifications and analysis use the layered specifications framework [21], to allow provable security for a system combining the delivery with evidences protocols, with higher and lower layer protocols.

References

1. Lacoste, G., Pfitzmann, B., Steiner, M., Waidner, M., eds.: SEMPER - Secure Electronic Marketplace for Europe. Volume 1854 of Lecture Notes in Computer Science. Springer-Verlag (2000)
2. Herzberg, A., Yoffe, I.: Layered Architecture for Secure E-Commerce Applications. In: SECRYPT'06 - International Conference on Security and Cryptography, INSTICC Press (2006) 118–125

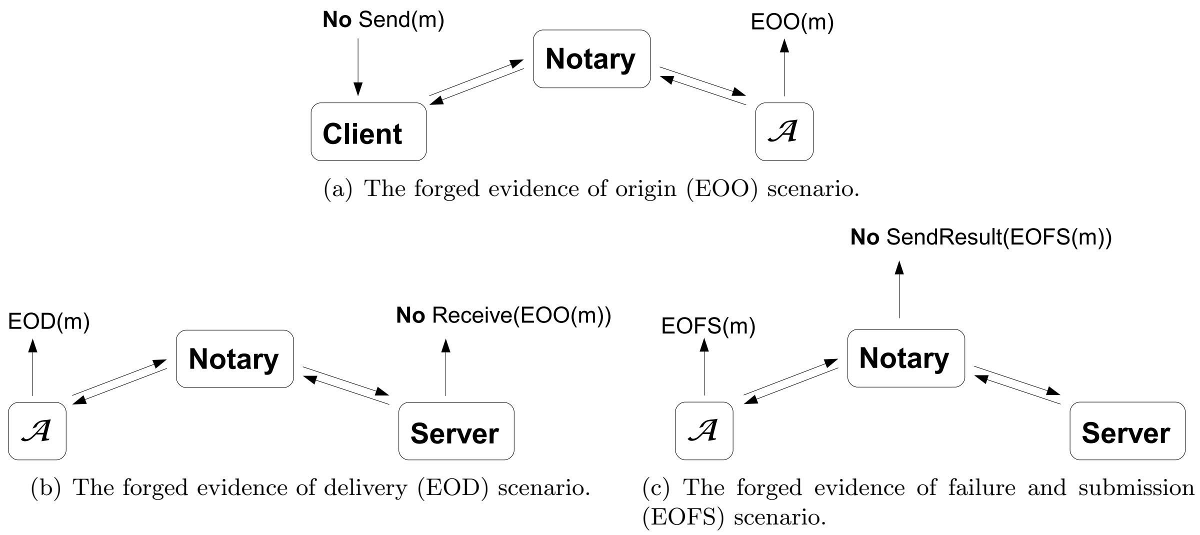
3. Asokan, N., Shoup, V., Waidner, M.: Optimistic Fair Exchange of Digital Signatures. IEEE Journal on Selected Areas in Communications **18** (2000) 593–610
4. Micali, S.: Certified E-Mail with Invisible Post Offices. Invited presentation at the RSA '97 conference (1997)
5. Micali, S.: Simple and Fast Optimistic Protocols for Fair Electronic Exchange. In: Proceedings of the 22nd Symposium Principles of Distributed Computing (PODC). (2003) 12–19
6. Pfitzmann, B., Schunter, M., Waidner, M.: Provably Secure Certified Mail. In: IBM Research Report RZ 3207 (#93253), IBM Research Division, Zurich (2000)
7. Kremer, S., Markowitch, O., Zhou, J.: An Intensive Survey of Non-repudiation Protocols. Computer Communications **25**(17) (November 2002) 1606–1621
8. Zhou, J.: Non-repudiation in electronic commerce. Computer Security Series. Artech House (August 2001)
9. ISO/IEC 10021: Information technology – Message Handling Systems (MHS). ISO/IEC (1999)
10. Agency, N.S.: Secure Data Network System : Message Security Protocol (MSP) (January 1996)
11. ISO/IEC 13888-1: Information Technology - Security Techniques - Non-Repudiation - Part 1: General. ISO/IEC (1997)
12. ISO/IEC 13888-3: Information Technology - Security Techniques - Non-Repudiation - Part 3: Mechanisms using asymmetric techniques. ISO/IEC (1997)
13. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Communications of the ACM **28**(6) (1985) 637–647
14. Garay, J., Pomerance, C.: Timed Fair Exchange of Standard Signatures. In: Financial Cryptography. Volume 2742 of LNCS., Springer-Verlag (2003) 190–207
15. Asokan, N., Schunter, M., Waidner, M.: Optimistic Protocols for Fair Exchange. In: CCS '97: Proceedings of the 4th ACM conference on Computer and Communications Security, New York, NY, USA, ACM Press (1997) 7–17
16. Zhou, J., Gollmann, D.: Observations on Non-repudiation. In: ASIACRYPT '96: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security, London, UK, Springer-Verlag (1996) 133–144
17. Park, J.M., Chong, E.K.P., Siegel, H.J.: Constructing fair-exchange protocols for E-commerce via distributed computation of RSA signatures. In: PODC '03: Proceedings of the 22nd Annual symposium on Principles of distributed computing, New York, NY, USA, ACM Press (2003) 172–181
18. Adams, C., Cain, P., Pinkas, D., Zuccherato, R.: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC Editor (2001)
19. Herreweghen, E.V.: Non-repudiation in SET: Open Issues. In: Proceedings of the 4th Conference on Financial Cryptography. (2000)
20. Nenadic, A., Zhang, N.: Non-repudiation and Fairness in Electronic Data Exchange. In: Proceedings of 5th International Conference on Enterprise Information Systems (ICEIS), Angers, France (2003) 55–62
21. Herzberg, A., Yoffe, I.: Layered Specifications Framework for Analysis of Secure Protocols. Cryptology ePrint Archive, Report 2006/398 (2006) `http://eprint.iacr.org/`.
22. Herzberg, A., Yoffe, I.: On Secure Orders in the Presence of Faults. In: Proceedings of Secure Communication Networks (SCN). Volume 4116 of LNCS., Springer-Verlag (2006) 126–140
23. Asokan, N., Shoup, V., Waidner, M.: Asynchronous protocols for optimistic fair exchange. In: Proceedings of the IEEE Symposium on Research in Security and Privacy. (May 1998) 86–99
24. Markowitch, O., Saeednia, S.: Optimistic Fair Exchange with Transparent Signature Recovery. In: FC '01: Proceedings of the 5th International Conference on Financial Cryptography, London, UK, Springer-Verlag (2002) 339–350
25. Markowitch, O., Kremer, S.: An Optimistic Non-repudiation Protocol with Transparent Trusted Third Party. In: ISC '01: Proceedings of the 4th International Conference on Information Security, London, UK, Springer-Verlag (2001) 363–378
26. Paterson, K.G., Yau, A.K.L.: Cryptography in Theory and Practice: The Case of Encryption in IPsec. In: EUROCRYPT, Springer-Verlag (2006) 12–29
27. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, London, UK, Springer-Verlag (2002) 83–107
28. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing Chosen-Ciphertext Security. Lecture Notes in Computer Science : Advances in Cryptology - CRYPTO 2003 (2003) 565–582

A Implementation of protocol and evidence validation

We specify the protocol in the context of a single evidence layer agreement Agr and denote by $v(Agr)$ a variable v obtained or stored in the context of such agreement. We would omit Agr, and use just v when the context is clear. We use *currenttime()* to denote the reading of the personal clock variable of a party running the protocol, and in addition we would use a boldface notation that implies that protocol events are invoked only if certain event strings are parsed and matched against state variables previously specified. That is, *on event(**Agr**,m,$\boldsymbol{\sigma}_\mathbf{m}$)* would mean that a protocol input *event* was parsed to a triple of values, and the first and last variable had been matched[1].

```
1:  on $EL.Init(1^k, addr, \alpha, \beta, \Delta_{comm}, V_{comm})$ :
2:    $status_{Clock} = Clock.Init(\alpha,\beta)$    // init clock service
3:    $status_{comm} = Comm.Init(addr, \Delta_{comm}, V_{comm})$    // init communication service
4:    $(sk,vk) = \mathcal{S}.Gen(1^k)$    // generate signing keys
5:    $(dk,ek) = \mathcal{CE}.Gen(1^k)$    // generate encryption keys
6:    $\Delta_{el} = (2 \cdot \Delta_{comm} + 2 \cdot \Delta_{comm} \cdot (1+\alpha)) \cdot (1+\alpha)$    // real-time bound on layer's response time
7:    if $(status_{Clock} \wedge status_{comm})$
8:      $EL.InitResult(vk, ek, \Delta_{el})$
9:    else
10:     $EL.InitResult($ false$)$
11: on $EL.OpenChannel(Agr,\rho)$ :
12:   if $(Agr.\rho = (addr,vk,ek) \wedge$    // check identity of self
13:       $\wedge\ (\alpha,\beta,\Delta_{comm},V_{comm}) = Agr.(\alpha,\beta,\Delta_{comm},V_{comm}))$    // check initialized variables validity
14:     $EL.OpenChannelResult($ true$)$
15:   else
16:     $EL.OpenChannelResult($ false$)$
```

Fig. 6. SELP initialization and channel establishment for evidence layer agreement *Agr*. Further protocol interactions are aborted if initialization or the following channel establishment do not succeed.



(a) The forged evidence of origin (EOO) scenario.

(b) The forged evidence of delivery (EOD) scenario.

(c) The forged evidence of failure and submission (EOFS) scenario.

Fig. 7. Forgery of evidences specifications.

[1] This notation would serve to reduce the number of explicit validations and verifications in the pseudo-code.

```
on $EL.Send(\boldsymbol{Agr},m')$:
  $t$ = $currenttime()$   // obtain local clock reading
  $m = \mathcal{CE}.Enc(S.ek,m',r)$   // encrypt message using randomness r
  if $(V_{comm}(m))$   // if message could be sent
    $\sigma_M = \mathcal{S}.Sign(sk,(Agr,\text{'M'},t,m))$   // sign message
    $Comm.Send(N.addr,(Agr,\text{'M'},t,c,\sigma_M))$   // send to notary
    $Clock.SetTimer(m,2 \cdot \Delta_{comm} + 2 \cdot \Delta_{comm} \cdot (1+\alpha)))$   // timer for notary's response
    $\mathbf{t}$=$t$,$\mathbf{m}$=$m$,$\mathbf{r}$=$r$
  else
    $EL.SendResult(Agr,\text{'FAIL'},\perp)$   // if message could not be sent
on $Comm.Receive(\boldsymbol{N.addr}, (\boldsymbol{Agr},\textbf{'R'},\boldsymbol{t},\boldsymbol{m},\sigma_R)$:
  if $(\mathcal{S}.Verify(S.vk, (Agr,\text{'R'},t,m),\sigma_R) \wedge$   // check recipient's signature
      $\wedge$ $timer(m)$ is on)   // check that message did not time-out
    $Comm.CancelTimer(m)$
    $EL.SendResult(Agr,(\text{'EOD'},t,m,(\sigma_M,\sigma_R)),(m',r))$   // supply evidence of delivery
on $Comm.Recieve(\boldsymbol{N.addr}, (\boldsymbol{Agr},\textbf{'F'},\boldsymbol{t},\boldsymbol{m},\sigma_F)$:
  if $(\mathcal{S}.Verify(N.vk, (Agr,\text{'F'},t,m),\sigma_F) \wedge$   // check notary's signature
      $\wedge$ $timer(m)$ is on)
    $Comm.CancelTimer(m)$
    $EL.SendResult(Agr,(\text{'EOFS'},t,m,(\sigma_M,\sigma_F)),(m',r))$   // supply evidence of failure and submission
on $Comm.WakeAlarm(\boldsymbol{m})$:
  $EL.SendResult(Agr,\text{'FAIL'},\perp)$
```

Fig. 8. SELP client role, $\rho = C$, pseudo-code for handling single, unique, message associated with previously specified evidence layer agreement *Agr*. The flow is aborted after first *EL.SendResult* invocation.

```
on $Comm.Recieve(\boldsymbol{C.addr}, (\boldsymbol{Agr},\textbf{'M'},t,m,\sigma_M))$:
  if $(S.Verify(C.vk, (Agr,\text{'M'},t,m), \sigma_M) \wedge$   // verify sender's signature
      $\wedge\ t \in [currenttime() - 2 \cdot \beta - \Delta_{comm} \cdot (1+\alpha), currenttime() + 2 \cdot \beta] \wedge$   // verify sender's time-stamp
      $\wedge\ V_{comm}(m))$   // verify message validity
    $Clock.SetTimer(m,2 \cdot \Delta_{comm})$   // timer for recipient's response
    $Comm.Send(S.addr, (Agr,\text{'M'},t,m,\sigma_M))$   // forward message
    $\mathbf{t}$=$t$,$\mathbf{m}$=$m$
on $Comm.Recieve(\boldsymbol{S.addr}, (\boldsymbol{Agr},\textbf{'R'},\boldsymbol{t},\boldsymbol{m},\sigma_R))$:
  if $(S.Verify(S.vk, (Agr,\text{'R'},t,m), \sigma_R) \wedge$   // verify recipient's signature
      $\wedge$ $timer(m)$ is on)   // verify recipient's response did not time-out
    $Clock.CancelTimer(m)$
    $\sigma_{RA} = S.Sign(sk,(Agr,\text{'RA'},t,m))$   // sign recipient's acknowledgement
    $Comm.Send(S.addr, (Agr,\text{'RA'},t,m,\sigma_{RA}))$   // send acknowledgement to recipient
    $Comm.Send(C.addr, (Agr,\text{'R'},t,m,\sigma_R))$   // forward recipient's response to sender
on $Clock.WakeAlarm(\boldsymbol{m})$:
  $\sigma_F = S.Sign(sk,(Agr,\text{'F'},t,m))$   // recipient timed-out, sign evidence of failure and submission
  $Comm.Send(C.addr, (Agr,\text{'F'},t,m,\sigma_F))$   // send evidence to sender
  $lastf$ = $currenttime()$   // remember last failure time lastf(Agr) for the current Agr agreement
```

Fig. 9. SELP notary role, $\rho = N$, pseudo-code for handling single, unique, message associated with previously specified evidence layer agreement *Agr*.

```
Server[$\rho = S$]: on $EL.AvailabilityRequest(\mathbf{Agr})$:
  $Comm.Send(N.addr,(Agr,\text{'AR'}))$ // send availability request to notary
  $cky = currenttime()$ // time readings used as unique cookie
  $Clock.SetTimer(cky,2 \cdot \Delta_{comm})$ // set timer for the response
  $\mathbf{cky}=cky$
Notary[$\rho = N$]: on $Comm.Recieve(\mathbf{S.addr}, (\mathbf{Agr},\mathbf{\text{'AR'}}))$:
  $I_A = [lastf(Agr),currenttime()]$ // pledge availability interval from last failure
  $\sigma_A = S.Sign(sk,(Agr,I_A))$
  $Comm.Send(S.addr, (Agr,\text{'A'},I_A,\sigma_A))$ // send signed message to server
Server[$\rho = S$]: on $Comm.Recieve(\mathbf{N.addr}, (\mathbf{Agr},\mathbf{\text{'EOA'}},[t_1,t_2],\sigma_{EOA})$:
  if $(\mathcal{S}.Verify(N.vk, (Agr,\text{'EOA'},[t_1,t_2]),\sigma_{EOA}) \wedge$ // verify notary's signature
        $\wedge$ timer(cky) is on) // verify notary's response did not time-out
     $EL.Receive(Agr,(\text{'EOA'},[t_1,t_2]),\sigma_{EOA},\perp)$ // supply evidence of availability
Server[$\rho = S$]: on $Comm.WakeAlarm(\mathbf{cky})$:
  $EL.Receive(Agr,\text{'FAIL'},\perp)$ // channel failure, as notary failed to respond
```

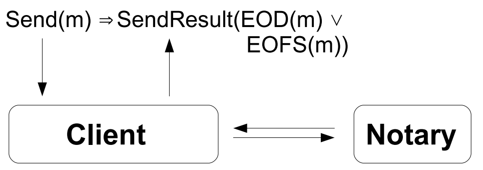
Fig. 10. SELP pseudo-code for availability, associated with previously specified evidence layer agreement *Agr*.

```
on $Comm.Recieve(\mathbf{N.addr}, (\mathbf{Agr},\mathbf{\text{'M'}},t,m,\sigma_M)$:
  if $(\mathcal{S}.Verify(C.vk, (Agr,\text{'M'},t,m),\sigma_M) \wedge$ // verify signature
        $\wedge\ t \in [currenttime() - 2 \cdot (\beta + \Delta_{comm} \cdot (1+\alpha))), currenttime() + 2 \cdot \beta] \wedge$ // verify sender's time-stamp
        $\wedge\ V_{comm}(m))$ // verify message validity
     $\sigma_R = \mathcal{S}.Sign(sk,(Agr,\text{'R'},t,m))$ // sign response
     $Comm.Send(N.addr, (Agr,\text{'R'},t,m,\sigma_R))$ // send response to notary
     $Clock.SetTimer(m,2 \cdot \Delta_{comm}))$ // timer for notary's response
     $\mathbf{t}=t,\mathbf{m}=m$
on $Comm.Recieve(\mathbf{N.addr}, (\mathbf{Agr},\mathbf{\text{'RA'}},\mathbf{t},\mathbf{m},\sigma_{RA})$:
  if $(\mathcal{S}.Verify(N.vk, (Agr,\text{'RA'},t,m),\sigma_{RA}))$ // verify notary's signature
     $(m',r) = \mathcal{CE}.Dec(dk,m)$ // decrypt message, and extract randomness used for encryption
     $Clock.CancelTimer(m)$
     $EL.Receive(Agr,(\text{'EOO'},t,m,(\sigma_M,\sigma_{RA})),(m',r))$ // supply evidence of origin
on $Comm.WakeAlarm(\mathbf{m})$:
  $EL.Receive(Agr,\text{'FAIL'},\perp)$ // channel failure, as notary failed to respond
```

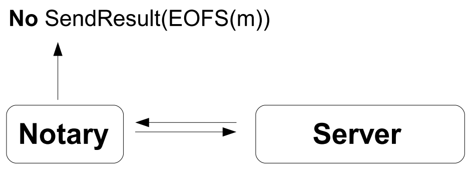
Fig. 11. SELP server role, $\rho = S$, pseudo-code for handling single, unique, message associated with previously specified evidence layer agreement *Agr*. The flow is aborted after first *EL.Receive* invocation.

```
Validate$(Agr,e,m,h)$:
  $(type,t,c,(\sigma_1,\sigma_2)) = e$
  if $((\mathcal{CE}.Enc(Agr.S.ek,m,h) \neq c) \vee (\neg Agr.V_{comm}(c)))$ return false
  case $type=EOFS$:
     return $\mathcal{S}.Verify(Agr.N.vk, (Agr,\text{'F'},t,c),\sigma_2) \wedge \mathcal{S}.Verify(Agr.C.vk, (Agr,\text{'M'},t,c),\sigma_1)$
  case $type=EOO$:
     return $\mathcal{S}.Verify(Agr.N.vk, (Agr,\text{'RA'},t,c),\sigma_2) \wedge \mathcal{S}.Verify(Agr.C.vk, (Agr,\text{'M'},t,c),\sigma_1)$
  case $type=EOD$:
     return $\mathcal{S}.Verify(Agr.S.vk, (Agr,\text{'R'},t,c),\sigma_2) \wedge \mathcal{S}.Verify(Agr.C.vk, (Agr,\text{'M'},t,c),\sigma_1)$
  case otherwise:
     return false
```

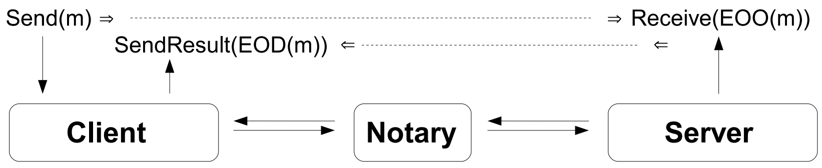
Fig. 12. Implementation of the `Validate` efficient algorithm for EOO,EOFS and EOD evidences validation for SELP. Validation of availability evidences (EOA) is similar and omitted for brevity.



(a) Client always gets evidences if client and notary are connected.



(b) Notary never declares server failure when server and notary are connected.



(c) Sent messages are delivered and both client and server obtain evidences when client, server and notary are connected.

Fig. 13. Liveness specifications.

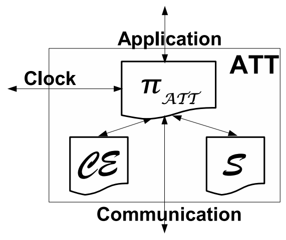
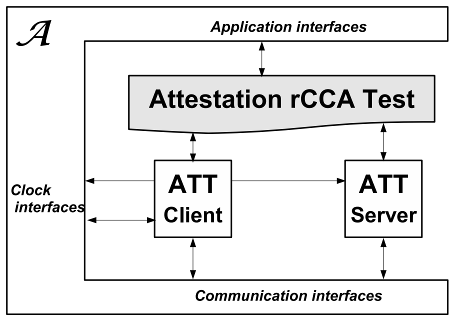
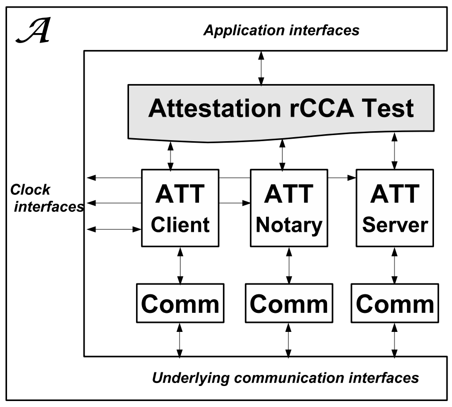


Fig. 14. Inner configuration of the attestation client and server blocks for Figures 15(a) and 15(b).



(a) Confidentiality (indistinguishability) test configuration with untrusted notary.



(b) Confidentiality (indistinguishability) test configuration with trusted notary.

Fig. 15. Confidentiality test configurations.