Towards Provably-Secure Timed E-Commerce: The Trusted Delivery Layer

Amir Herzberg
Computer Science Dept.
Bar Ilan University
hPost://amir.herzberg.name

Abstract

Certified exchange of messages is an essential mechanism for e-commerce; the timing aspects (timeouts and timestamps) are very important for practical applications. However existing formal methods for security analysis assume simplified completely synchronous or completely asynchronous models, and cannot deal with the timing aspects of these (and other e-commerce) protocols. We present model for realistic, Δ-synchronized adversarial settings. We then present a simple, efficient and provably-secure protocol for certified, time-stamped message delivery, providing precise guarantees of delay and timestamps. Our model and analysis use concrete (rather than asymptotic) notions of security.

Keywords: *secure electronic commerce; non-repudiation; timestamp; certified delivery; certified mail; certified e-mail; notarized delivery; notarization; e-banking; contract signing; timestamping*

1. Introduction

Trustworthy, agreed-upon method of communication is an essential tool for any business relationship, online and offline. Indeed, essentially every contract includes a provision stating the agreed-upon mechanisms of communication, and what will constitute a sufficient proof of submission and delivery of messages between the parties. In traditional contracts and where risks are substantial, contracts typically specify a certified mail service operated by a trustworthy postal system (`Post`). We show how such services can be provided electronically, i.e. how to implement a certified mail (messaging, e-mail) system.

Contributions of our model. Our model continues the work on secure reactive systems of [PSW00a, PSW00b, PW01], with the following contributions:

- We allow adversarial scheduling with realistic, limited synchronization assumptions (*Δ-synchronization*), unlike previous analytical models which were purely synchronous [PSW00] or purely asynchronous [ASW00, PW01].
- We use concrete security analysis following [BKR94,BCK96], rather than asymptotic security analysis as in the previous works on secure reactive systems. We consider also concrete length of inputs, outputs and messages, since these can have significant impact on security and performance of protocols.
- Our model is simpler than the previous models, due to our focus on adversary-driven interactions between machines, which is the common approach to security analysis, and on limiting our attention to single thread of computation and non-adaptive adversaries.

Contributions of our protocol. There has been extensive amount of research on secure certified (electronic) mail, and the related Trusted Third Party protocols of fair exchange, contract signing and timestamping; we consider all these to be variants[1] of the *trusted delivery problem,* and refer to the Trusted Third Party as Post. Our protocol extends Protocol F of [ASW00], and shares many of its properties; in

[1] The variants differ mainly regarding confidentiality of the messages (many works require this but we consider it separately in section 4.1, independently of the trusted delivery protocol), timeliness (which we provide), what the recipient provides (in our case: a response and signature over response and message from the sender; other definitions, esp. of fair exchange, allow more general definition of what the recipient should provide).

particular it is optimistic – i.e., the Post 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 timestamp. Previous works assumed perfect synchrony, or did not provide timestamp 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 Post tries to provide services incorrectly, this can be proven based on the affidavits signed by the Post or detected in real time. Note that *accountability implies a visible Post* server, i.e. the identity of the post can be inferred from the affidavits produced by the protocol. We believe that for most e-commerce applications, e.g. e-banking, accountability is crucial while invisibility of the Post is not; however most works do not provide accountability (sometimes, in order to provide invisible Post [M97,M03]). One notable exception is Protocol F of [ASW00].
- Our protocol is simple, practical and efficient, yet proven secure. The efficiency gain is substantial compared to rigorously proven protocols (e.g. [PSW00]).

Related Protocols. There have been many works on fairness and non-repudiation services, we only provide partial and concise comparison; for more bibliography of non-repudiation protocols, see [KMZ02,Z01].

An underlying delivery layer for secure e-commerce applications is part of the architecture of [LPSW00], who 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 [X.400,MSP96], and ISO standards [ISO13888-1,ISO13888-3].

Much of the research on fair delivery focuses on avoiding the use of a Post, by probabilistic and/or gradual exchange, as in [G82, EGL85]; these works are applicable only to special applications, mainly due to their high overhead and probability of failure; and certainly they cannot provide timestamping services.

Our work follows the `optimistic` approach, i.e. involves the Post only to handle exceptions. Many works study optimistic fair delivery protocols, e.g. see [ASW97, ASW00, KMZ02, M97, M03, PSW00, Z01, ZG96, PC*03]; these works avoid the involvement of the Post in typical, fault-free executions[2].

The timeliness properties of the protocol imply, in particular, that the `proofs` produced include a digitally signed *time-stamp.* The timestamp allows the receiver (sender) to prove to a third party, e.g. judge, 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 (respectively, receiver), except if the receiver fails, in which case the timestamp is by the Post (on the POS). Some applications may require a timestamp on a signature by a trusted third party (not the signer) to enable long-lasting signatures with possibly exposure of (old) keys; we present a simpler version of our protocol, Simple Trusted Delivery Protocol, which could be used in these cases. This is more efficient than having the parties contact a Time Stamping Authority, as with the standard Time Stamp Protocol (TSP) [RFC3161].

In most of this work, we do not require the trusted delivery layer to provide confidentiality. Confidentiality against eavesdroppers which are not one of the parties in the protocol can be achieved independently of our protocol, by using appropriate security mechanisms in the networking layers, such as SSL/TLS or IP-Sec, as illustrated in **Figure 1**. However, many of the previous works on certified e-mail provide confidentiality also against the Post and the receiver. Confidentiality against the receiver implies that the receiver should be able to view the plaintext message only if the sender receives the Proof Of Delivery, i.e. is an additional fairness requirement. Confidentiality against the Post implies that the Post is not aware of the contents of the message sent, even if it is involved in the delivery; only if the parties need to present their proofs (e.g. to

[2] Since the Post is not involved `online` for typical, fault-free transactions, these protocols are sometimes called `offline`; we prefer the (more common) term `optimistic`.

court) they may have to expose the contents of the message (to the court); this feature is also provided by many timestamping systems. However, adding confidentiality to the formal specifications (and analysis) of trusted delivery introduces substantial complexity (e.g. see in [PSW00). We believe that we present a much more elegant solution in Section 4.1. Namely, confidentiality is ensured by applying any *Commit-Encrypt* scheme [GH03], such as standard RSA, outside (`above`) the trusted delivery layer (see **Figure 1**).

Organization. In Section 2, we present our protocols and specifications. In Section 3 we present the *Δ-synchronized* reactive systems model, and use it to define formal specifications for trusted delivery and to analyze our protocols. In Section 4 we discuss possible extensions and implementation issues, and in section 5 we conclude.

2. Protocols and requirements

We propose to provide support for common trusted-delivery services in a Trusted Delivery Layer, which will be placed between the e-commerce applications (application layer) and the networking protocols stack, on top of a reliable transport layer, typically TCP. Confidentiality can be provided by Transport Layer Security (TLS/SSL) [R00,RFC2246] or Internet Protocol Security (IP-sec) [RFC2411], or independently of the Trusted Delivery layer, as we sketch in Section 4.1. See Figure 1.

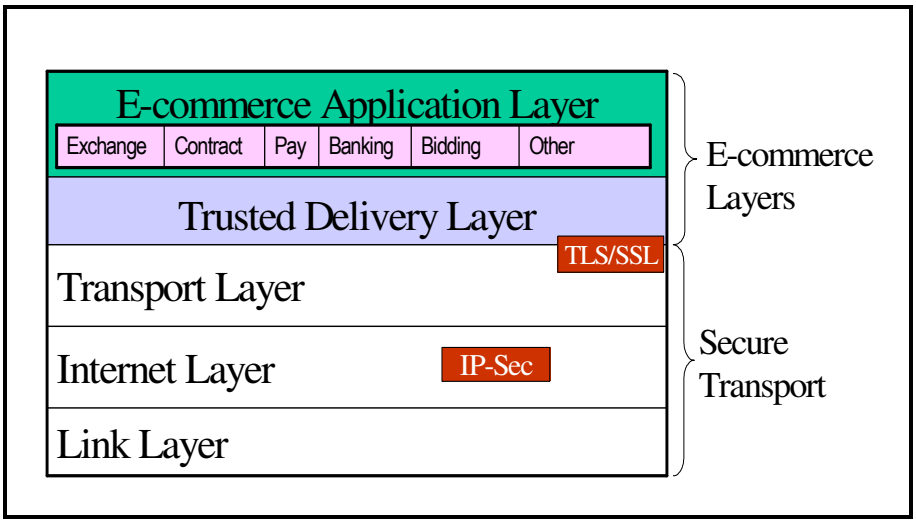

Figure 1: Secure E-Commerce and Networking Layers

In this section we present, informally, a simple trusted delivery protocol (STDP). This protocol is very natural, and indeed follows a 'common sense' procedure which is appropriate when using either physical security (e.g. handwritten signatures) or cryptographic security mechanisms (e.g. digital signatures). From STDP, we define specifications for the TDL. We then present our final protocol, the Optimistic Trusted Delivery Protocol (OTDP), as a simple enhancement of STDP. Before we present STDP, we first present our assumptions regarding the underlying communication and clock synchronization services.

2.1. Communication and Clock Assumptions

We begin by assuming that the underlying networking layers ensure reliable communication between non-faulty parties, with maximal delay Δ, using addresses known to the parties. Notice that we permit message reordering and injection; the protocol will provide its own integrity mechanisms (signatures) to deal with these issues. Also, in reality, the network may sometimes fail to deliver messages, e.g. due to denial of service attacks. However, since one of the goals identified above is to allocate responsibility for faults, including network faults, we consider such faults as belonging to either sender or recipient (never to the Post). We assume that the runs of the system has the following limited, realistic synchrony and transmission delay and rate properties.

Definition 2-1 Parties *u,v* are *[Δ,ρ]-Sync connected* in run *r* if the following properties hold:

1. Party u (and v) knows its network (e.g. IP) address $addr[u]$ (respectively, $addr[v]$).
2. **Maximal message delay Δ**: every message sent by u at time t to v, is received within $(t,t+\Delta]$.
3. **Maximal offset Δ**: party u has a monotonously-increasing clock variable c_u such that at every time t holds $|t-c_u[t]|<\Delta$, i.e. c_u is always within offset of at most Δ from real time.
4. **Maximal wakeup delay Δ**: party u can invoke, at any time t, a function $sleep(T)$ where $T>0$. As a result, u will be invoked by a *wake* call during $[t+T,t+T\Delta)$.
5. **Transmission rate Δ/ρ:** every party can send a message (at least) once every Δ/ρ time units; we call ρ the *guaranteed minimal rate* or simply the rate. For our purposes, it is sufficient[3] to assume $\rho>4$.

As a simplification, we used the same bound Δ for message delay, offset and wakeup delay. In reality, the bounds may differ, and are assured by different mechanisms. The message delay bound is typically provided by the TCP timeout timers, while the offset and wakeup delay bounds are typically ensured by running a clock synchronization protocol such as the Network Time Protocol [M90]. Notice that the bounds set by TCP and NTP work only when the processors operate correctly and have an operational connection to the network; however as discussed above, if this does not hold for a party, then it is reasonable to consider that party responsible for the failure of the protocol (i.e. consider the fault as occurring in the party). Secure versions of these protocols may also be used.

2.2. Simple Trusted Delivery Protocol (STDP)

In this section we present a simple trusted delivery protocol (STDP). This protocol can be implemented using either physical security (envelops and seals/stamps) or cryptographic security. STDP involves the Post office in every execution, including typical, faults-free executions; we later present the Optimistic Trusted Delivery Protocol, which involves the Post only in atypical scenarios. Figure 2 below shows a typical execution of STDP.

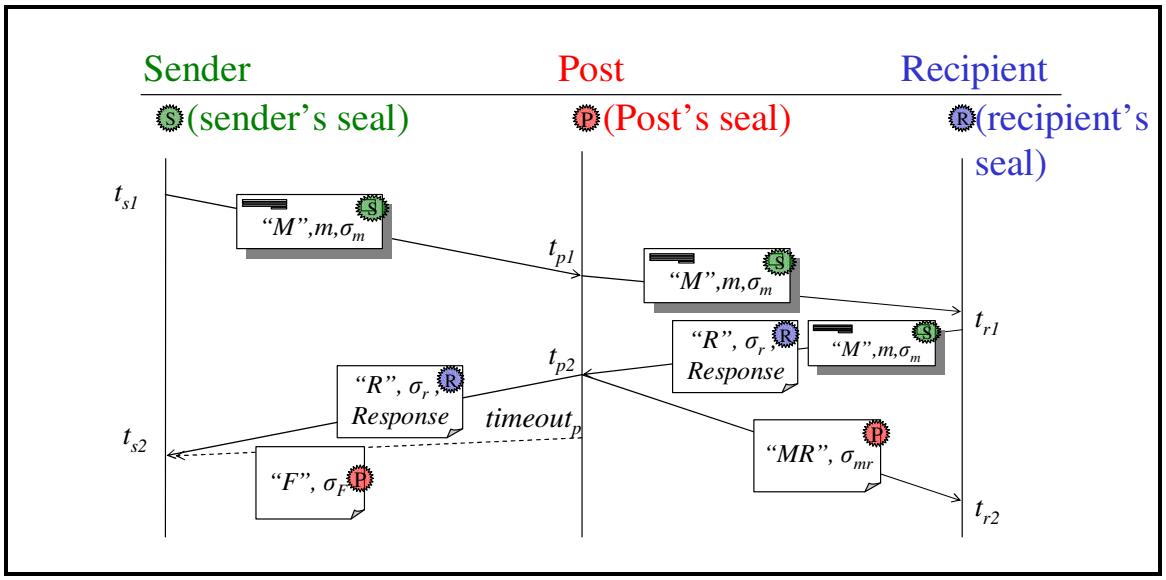

Figure 2: Simple Trusted Delivery Protocol (STDP)

We now explain the steps of the protocol. We describe, for each step, implementation based on physical security and implementation based on cryptography. The first two initialization steps are not in the figure.

1. The application initializes the protocol, providing the security parameter k and the maximal message length MTU. The TDL generates private signature key s and public verification key v and returns v and the network address $addr$ to the application.

[3] In practice, the limit on the transmission rate (time between sending two messages) is usually very small, and it is often ignored in analysis of protocols and distributed algorithms. Since we allow the adversary to control scheduling, including inter-message delay, we cannot completely ignore it (but we expect ρ to be quite large). Assuming larger value of ρ may be needed to support multiple simultaneous deliveries, and will slightly improve the bounds we prove.

2. The application forms and provides to the TDL the agreement *a* containing the verification keys *a.s.v, a.r.v* and *a.p.v* and the network addresses *a.s.addr, a.r.addr* and *a.p.addr* of the parties (Sender, Recipient and Post, respectively, using dot notation).
3. At t_{s1}, the sender accepts from the higher layer a message *m* for delivery to the recipient. The sender signs *m* together with `timestamp interval` $I=[c_s(t_{s1})-5\Delta, c_s(t_{s1})+9\Delta]$, calculated below, during which the message is to be delivered. When using digital signatures, the signature is $\sigma_m=Sign_{a.s.sk}(\text{“M”},a,m,I)$. (We discuss support for confidentiality later on). It then sends *m, I* and[4] the signature/timestamp σ_m to the Post and waits up to time-out bound $TO\_t_{s2}=(5+1/\rho)\Delta$ for response or failure affidavits (see calculation of TO_t_{s2} below).
5. At $t_{p1}<t_{s1}+\Delta$, the post office receives the message *m*, interval *I* and signature/timestamp σ_m, sends them[5] to the recipient, and waits for response (expected within the maximal round trip delay of 2Δ).
6. At $t_{r1}<t_{s1}+\Delta$, the recipient receives *m, I* and σ_m from Post and sends back *response* and a signature σ_R on *response*, and waits (up to 2Δ) for the affidavit σ_{mr} from the Post. When using digital signatures, the signature on the response is $\sigma_r=Sign_{a.r.s}(\text{“R”},a,m,response,I)$.
7. At $t_{p2}<t_{r1}+\Delta$, the post office receives the signed response from the recipient, and validates it. It then sends to the recipient affidavit σ_{mr} for the message origin and response. When using digital signatures, the affidavit is $\sigma_{mr}=Sign_{a.p.s}(\text{“MR”},a,m,response,I)$. Finally, it forwards the response to sender. At this point the post office completed its operation in the protocol.
8. If by $t\geq t_{p1}+2\Delta$ the post office did *not* receive the signed response, or upon receiving an invalid response (e.g. not signed correctly), the post office sends the sender a (signed) failure affidavit σ_F. When using digital signatures, the affidavit for the failure is $\sigma_F=Sign_{a.p.s}(\text{“F”},a,m,I,c_p(t))$.
9. At $t_{s2}<min(t_{s1}+\Delta,t_{s2})+3\Delta$, the sender receives either the signed response or the failure affidavit, and terminates. Similarly, at $t_{r2}<t_{r1}+2\Delta$, the receiver receives the affidavit σ_{mr} identifying the message and response, signed by the Post.
10. If the sender or recipient detects time-out for any event above, they abort with appropriate alarm of failure in communication with Post.

The Post office ensures fairness and availability:

- Fairness: the post office sends the affidavit to the recipient and the (signed) response to the sender at the same step (7).
- Availability: we assume that the post office is always operational and connected to the parties, i.e. communication failures occur only at the sender and recipient. Hence, if the sender is not faulty, it should receive receipt of submission and later either receipt of delivery or failure report; and if the recipient is non-faulty then the sender should never receive a failure report.

In reality, the post office may fail – either due to benign (accidental) failure, or due to malicious corruption. In particular, the post office may fail to operate or to communicate with one or both parties. The sender detects when it does not receive in time a receipt of submission, or later on, when it does not receive in time receipt of either delivery or failure. Similarly, the recipient detects if it does not receive the message, within reasonable time after sending the response. However, the parties cannot `prove` these failures of the post office to a third party; their only recourse is to raise appropriate *alert* immediately upon detecting the failure (which is shortly after the failure occurred).

A faulty post office may also send failure report to the sender, without properly requesting receipt for delivery from the recipient. To protect against this threat, the Post needs to send an *affidavit of availability* to the receiver, for periods where no fault was detected. The Post could send these periodical reminders once every Δ time units, e.g. by installing an appropriate timer. Upon waking up, say at time t_A, the Post sends[6] to the Recipient *affidavit of availability* $\sigma_F=Sign_{a.p.sk}(\text{“A”},a,(c_p(t_F), c_p(t_A)))$ where t_F is the last time

[4] We did not assume message recovery property from the signature scheme, but of course could it when available.
[5] In some applications, it is useful to split the message *m* to two parts: message cover *m.cover* which is sent at this point, and message body *m.body* which is sent by the Post to the receiver only when the Post receives the response from the receiver. In these cases it may be desirable also to send the signature σ_m later; this requires authentication of the messages from Post to Recipient.
[6] When the Post sent recently an “MR” affidavit it may be able to skip sending the “A” affidavit but we’ll ignore this.

that the Post sent failure affidavit for this Recipient (or the beginning of the run, if the Post did not detect failure for this Recipient so far).

A faulty Post could still send conflicting affidavits of failure and of availability, and the parties would not be able to detect this in `real time`. However, upon dispute resolution, this fraud will be evident from the conflicting signatures, and the Post would therefore be held accountable.

2.2.1. Determination of Time-Outs and interval *I* in STDP

Our description of STDP left open the length of the interval *I* used in the protocol; we also did not explain the time-out values. We now show how these values are calculated.
We first notice that the maximal round-trip delay is 2Δ. This explains the time-out limit for the affidavits from the Post by the sender and recipient, and for the response.

Next, we calculate the time-out bound for t_{s2}, denoted TO_t_{s2}. This timeout should be sufficient to receive either response or failure affidavit, which would be sent no later than $timeout_p$. If a response is sent, then the Post first sends the "MR" affidavit to the Recipient; the maximal delay of the response or failure affidavit to the Sender is therefore $\Delta(1+1/\rho)$ (maximal delay Δ plus inter-transmission delay Δ/ρ). It remains to bound $timeout_p$ using values known to the sender. The sender knows that $t_{p1} \le t_{s1}+\Delta$; at this point Post begins waiting for the response, which should arrive within 2Δ. But due to the possible wakeup delay, we have $t_{p1}+2\Delta<timeout_p<t_{p1}+3\Delta$. Hence, $\boldsymbol{TO\_t_{s2}=(5+1/\rho)\Delta}$. (Recall that ρ is typically large.)

The last time-out bound is for affidavits of availability, denoted TO_A. This time-out is set by the Recipient upon receiving an affidavit of availability from Post at some time *t*. The Post should send the next affidavit of availability within at most 2Δ, and it could take up to one additional Δ for this affidavit to arrive at the Recipient. Therefore, $TO_A=3\Delta$.

It remains to calculate the interval *I* as set by the Sender, and the bounds on it I_P, I_R validated by the Post office and Recipient, respectively. The receiver validates, at t_{r1}, that *I* contains the maximal time of delivery $t_{r2} \le t_{r1}+2\Delta$. To validate, the receiver notes that $c_r(t_{r1})+\Delta>t_{r1}>c_r(t_{r1})-\Delta$. Hence it validates that $\boldsymbol{I \supseteq I_R \equiv [c_r(t_{r1})-\Delta,\ c_r(t_{r1})+3\Delta]}$.

When the Post receives the message *m* and the signature $\sigma_m=Sign_{a.s.sk}(\text{"Origin"},a,m,I)$, at t_{p1}, it needs to confirm that the Recipient will find the interval *I* in the message acceptable, i.e. $I \supseteq I_R$. Since each clock has maximal offset of Δ from real time, then the maximal bias between two clocks is 2Δ, and in particular $c_p(t_{p1})-2\Delta<c_r(t_{p1})<c_p(t_{p1})+2\Delta$. Also, since the delay is at most Δ, it follows that $c_r(t_{p1})<c_r(t_{r1})<c_r(t_{p1})+\Delta$. Hence Post confirms that $\boldsymbol{I \supseteq I_P \equiv [c_p(t_{p1})-3\Delta,\ c_p(t_{p1})+6\Delta]}$. Similarly, $c_s(t_{s1})+3\Delta>c_p(t_{p1})>c_s(t_{s1})-2\Delta$; hence to ensure that the validations by Post and Recipient will be successful, the Sender uses $\boldsymbol{I=[c_s(t_{s1})-5\Delta,\ c_s(t_{s1})+9\Delta]}$.

2.2.2. Validation of Affidavits in STDP

A central element of STDP (and any trusted delivery protocol) is the mechanism of 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 it submitted the message/request on time; that the message was delivered to the destination, who made a specific response, on given 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 affidavits (statements, receipts) during the run.

It is critical that the parties have a clear agreement on the process to validate affidavits (and thereby claims). We next define a validation function *V* for the cryptographic implementation of the STDP (the 'physical' implementation is obvious). The first parameter of the validation function is the type of affidavit: "R" and

"F" for the sender, and "M", "MR" and "A" for the recipient. The validation is similar in all cases: a validation of the signature of the signer of the corresponding affidavit (sender for affidavit of origin, recipient for affidavit of response, and Post for affidavits failure and availability).

$$V_{STDP}("MR", a, m, response, I, \sigma_m, \sigma_{mr}) = \begin{cases} True\ if\ Verify_{a.s.v}(<"M", a, m, I>, \sigma) \wedge Verify_{a.p.v}(<"MR", a, m, response, I>, \sigma) \\ False\ otherwise \end{cases}$$

$$V_{STDP}("R", a, m, response, I, \sigma) = \begin{cases} True\ if\ Verify_{a.r.v}(<"R", a, m, response, I>, \sigma) \\ False\ otherwise \end{cases}$$

$$V_{STDP}("F", a, m, I, \sigma) = \begin{cases} True\ if\ Verify_{a.p.v}(<"F", a, m, I>, \sigma) \\ False\ otherwise \end{cases}$$

$$V_{STDP}("A", a, I, \sigma) = \begin{cases} True\ if\ Verify_{a.p.v}(<"A", a, I>, \sigma) \\ False\ otherwise \end{cases}$$

The last parameter to the validation function V is the affidavit (`proof`). The identities of the parties, and their public keys, are given in the agreement a; e.g. the verification key of the Post is $a.p.vk$. The affidavit for recipient requires two signatures σ_m (by sender) σ_{rm} (by Post); the Post's signature also contains the response.

2.3. Application Interface and Requirements

We identify the following types of events between the application and the Trusted Delivery Layer (TDL):

1. **Initialization events:**
   - *init:* Application initializes TDL, specifying security parameter k and maximal message length *MTU*.
   - TDL outputs the (public) verification key v and address *addr.*
   - Application inputs the agreement a, containing the verification keys *a.v.s, a.v.r, a.v.p* and addresses *a.addr.s, a.addr.r, a.addr.p*, for Sender, Recipient and Post, respectively.
2. **Input events:** Application provides the message (at sender) and response (at recipient).
3. **Output events:**
   - **In the recipient:**
     - A message together with "MR" affidavits signed by the sender and/or Post.
     - Affidavit of availability signed by Post, for a given time interval.
     - An *alert*, indicating timeout while waiting for the affidavits.
   - **In the sender:**
     - Affidavit of (delivery and) response ("R"), signed by Recipient.
     - Affidavit of Failure for the submitted message, signed by Post.
     - An *alert*, indicating timeout while waiting for the affidavits.
   - The adversary may also output various affidavits (to demonstrate attack).

We now identify the requirements of applications using the Trusted Delivery Layer (TDL). In this version, our specification of the requirements is a bit informal.

Definition 2-2 We say that run z is a *Δ,T execution of TDL* with sender *s,* receiver *r* and Post office *p*, or that *TDL[Δ,T](z,<s,r,p>)=Ok,* if the following holds:

- **Efficient initialization:** Each party (sender, recipient and Post) process *init* and *agreement* invocations by the application in less than Δ time units.
- **Affidavits are valid.** Whenever a non-corrupt party outputs an affidavit, then the result of applying the validation function to it is always `valid`.

- **Valid affidavits imply event.** If the adversary (or any party) outputs a valid affidavit of…
  - **"MR" affidavit (with message *m*, response *r*),** and the Sender is non-corrupt, then the sender originated *m* during the indicated time interval. If the post and sender are non-corrupt, then the receiver sent response *r*.
  - **"R" affidavit (with message *m*, d response),** and the recipient is non-corrupt, then the recipient delivered the indicated message from the sender, and made the indicated response, during the indicated time interval.
  - **"F" affidavit (with a message *m*),** and the Post is non-corrupt, then the sender originated the message during the indicated time interval but the recipient failed to respond.
  - **"A" affidavit (with time interval),** and the Post and Sender are non-corrupt, then the recipient provided receipts of delivery to all requests from it during the indicated time period.
  - **Both "F" and "A" or "MR" for overlapping periods signed by Post[7]:** post is corrupt.
- **Alerts indicate a fault:** whenever a party outputs an *alert*, then either this party or the Post is corrupted (faulty). Notice that communication failures are also considered as faults of the parties.
- **Delivery time guarantees:**
  - The time intervals specified in affidavits produced (by non-faulty parties) are less than *T*.
  - At most *T* time units after a non-corrupt sender is invoked to deliver a message, it outputs an *alert* or (valid) affidavit of response or failure.
  - If the recipient is not corrupted, then it either produces "A" affidavits for the entire duration of the run (after receiving the response from the TDL), or if there is no "A" affidavit containing some time *t*, then the recipient issues *alert* before *t+T*.
- **Fairness:**
  - If the recipient is non-faulty and the sender outputs valid "R" affidavit, then within the time interval specified in the affidavit, the recipient outputs the message and (valid) "MR" affidavit, or *alert.*
  - If the sender is non-faulty and the recipient outputs a valid "MR" affidavit, then within the time interval specified in the affidavit, the sender outputs a valid "R" affidavit or *alert*.

2.4. Optimistic Trusted Delivery Protocol

The Optimistic Trusted Delivery Protocol tries first to exchange the message and affidavits of message and of response directly between the sender and receiver. If this fails, it continues like STDP; see **Figure 3**. Specifically, OTDP operates as follows:

1. As in STDP, initially the sender, recipient and post agree on agreement *a* containing verification keys *a.s.v, a.r.v* and *a.p.v* and addresses *a.addr.s, a.addr.r, a.addr.p*.
2. At t_{s0}, the sender accepts a message *m* for delivery to the recipient. The sender signs *m* together with `timestamp interval` $\boldsymbol{I=[c_s(t_{s0})-3\Delta, c_s(t_{s0})+13\Delta],}$ as calculated below, during which the message is to be delivered. When using digital signatures, the signature is $\sigma_m=Sign_{a.s.s}(\text{"M"},a,m,I)$. It then sends *m, I* and σ_m to the Recipient and waits 2Δ for the (signed) response.
3. At $t_{r0}<t_{s0}+\Delta$, the Recipient receives the message *m*, interval *I* and signature/timestamp σ_m. It returns to the sender a *response* and a signature $\sigma_r=Sign_{a.r.s}(\text{"R"},a,m,response,I)$.
4. At $t_{s0F}<t_{r0}+\Delta$, the Sender receives *response,* σ_r from the Recipient, sends back a signed affidavit $\sigma_{mr}=Sign_{a.s.s}(\text{"MR"},a,m,response,I)$, and terminates. Upon receiving this affidavit the recipient validates it, delivers it (and the message) to the higher layer, and terminates.
5. If the Sender does *not* receive the response from the receiver by the timeout (scheduled for $t_{s0}+2\Delta$), then it concludes that there is a failure, and continues like in STDP.
6. If the Recipient does *not* receive the signed σ_{mr} affidavit by the timeout (scheduled for $t_{r0}+2\Delta$), then it concludes that there is a failure, and continues like in step t_{r1} of STDP. However, note that the Recipient needs to send to the Post the message *m* and interval *I*, since the Post, in this scenario, did *not* receive these values from the Sender.

[7] An "A" or "MR" affidavit signed by Sender remain valid (even if there is also an "F" affidavit).

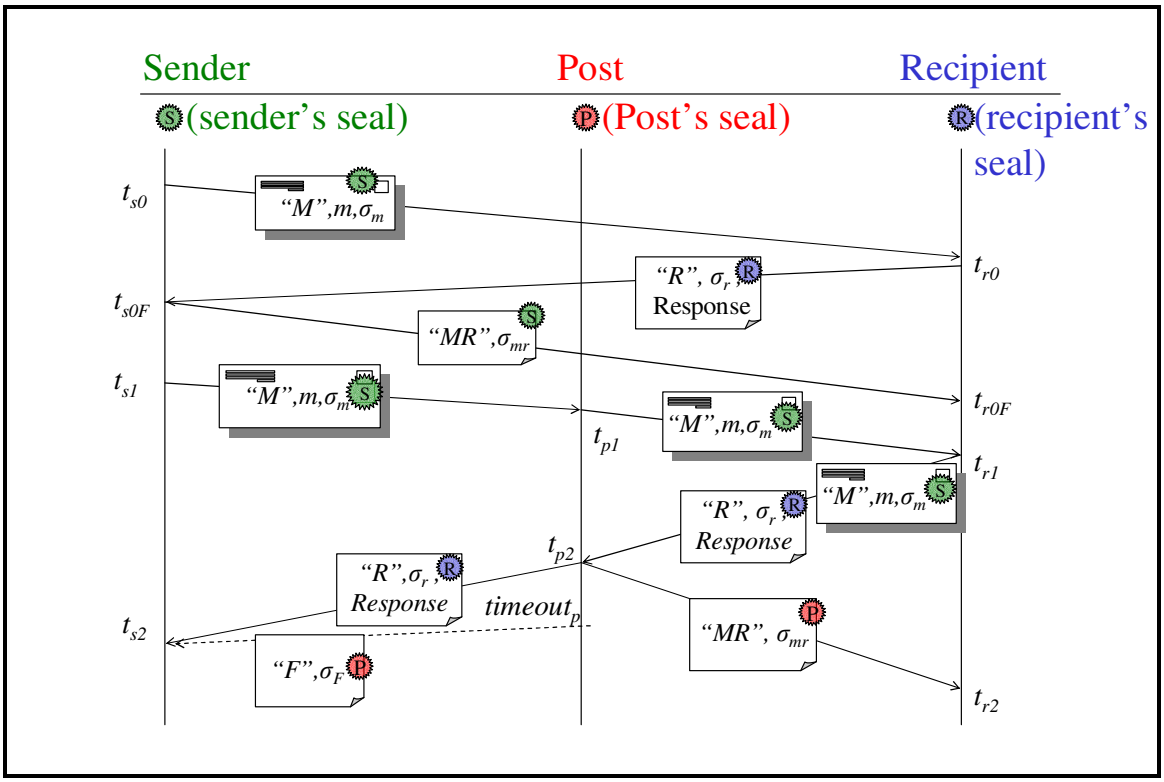

Figure 3: Optimistic Trusted Delivery Protocol (OTDP)

In addition, we OTDP runs an `optimistic availability affidavit sub-protocol`, which provides the Recipient with affidavits of availability ("A" affidavits) from either Post (as in STDP) or Sender. This mechanisms is slightly more complicated than the trivial one described for STDP, since we want to involve the Post only in failure scenarios. This simple sub-protocol is illustrated in Figure 4, and defined as follows. In this pseudo-code, let *c* be the latest reading of the local clock of the party.

Optimistic Availability Affidavit Sub-Protocol of OTDP:

Sender: On Wake at time *t*: { if no failure so far: send $Sign_{a,s.s}$("A",*c*); sleep (*Δ*)}

Post: On receiving "Req A Affidavit" at time *t:* if no "F" affidavit sent yet, send $Sign_{a,p.s}$("A",*c*);

Recipient:

Every *Δ* do:

{ Let t_s be the latest value in valid "A" affidavit received;
if t_s<*c*-6*Δ* then Alert("Failed to receive A affidavit");
else if t_s<*c*-3*Δ* then send "Req A Affidavit" to Post;
}

On receive of valid "A" affidavit: deliver it to higher layer.

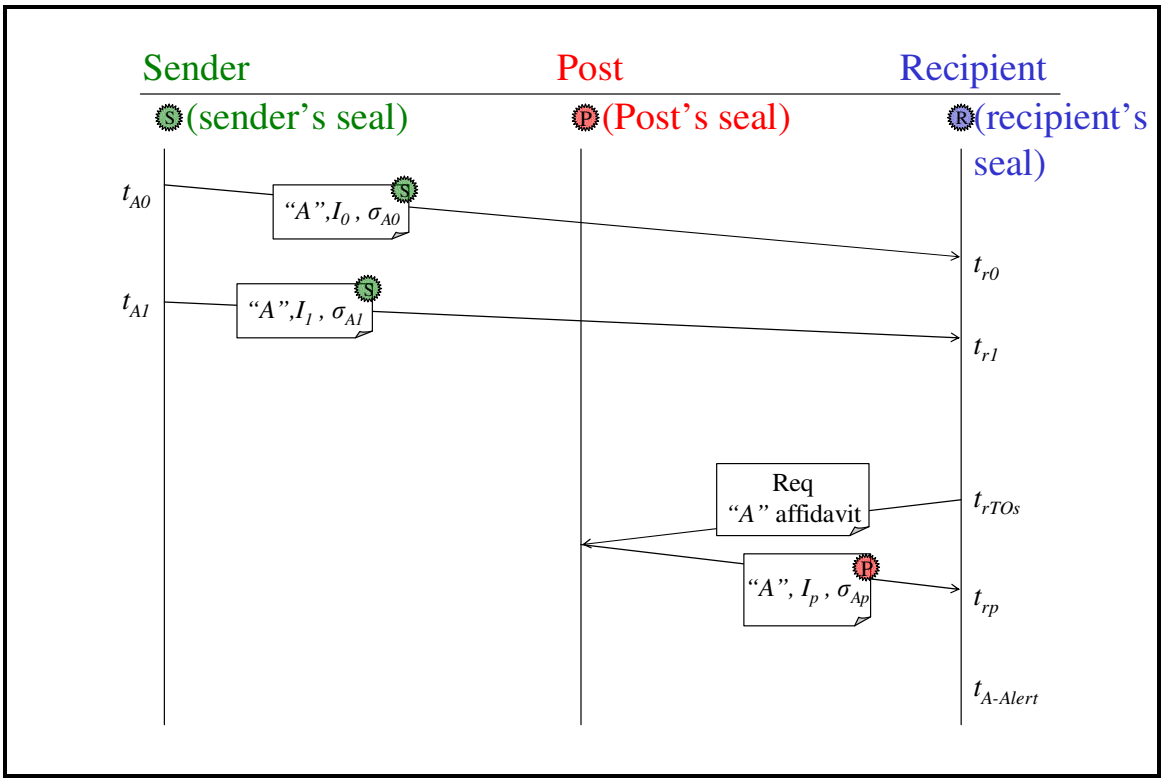

Figure 4: Optimistic Availability Affidavit Sub-Protocol of OTDP

2.4.1. Determination of interval *I* in OTDP

We now calculate the interval I in OTDP, as set by the Sender. Like in STDP, the Recipient and Post validate the received interval I is Ok when they receive the first message of the protocol, but notice that in OTDP this can happen at two points for both of them: for the Recipient, either at t_{r0}, upon receiving the message directly from the Sender, or at t_{r1}, upon receiving the message from the Post; and for the Post, either at t_{p1}, upon receiving the message from the sender, or at t_{p2}, upon receiving the message (and response) from the Receiver. The calculation is therefore a bit more complex than that for STDP.

Validation by the Recipient at t_{r1} (when receiving message from Post) and by the Post at t_{p1} is exactly like in STDP. Namely, The receiver validates at t_{r1} that $\boldsymbol{I \supseteq I_{R1} \equiv [c_r(t_{r1})-\Delta,\ c_r(t_{r1})+3\Delta]}$, and the Post validates at t_{p1} that $\boldsymbol{I \supseteq I_{P1} \equiv [c_p(t_{p1})-3\Delta,\ c_p(t_{p1})+6\Delta]}$. To compute I at the sender, we need to bound $c_p(t_{p1})$ and $c_r(t_{r1})$ using $c_s(t_{s0})$. Note first that $t_{s1}<t_{p1}<t_{r1}$ and $t_{r1}<t_{p1}+\Delta<t_{s1}+2\Delta$; and the maximal bias between any two clocks is 2Δ. Finally, due to the maximal wakeup delay, $t_{s0}+2\Delta<t_{s1}<t_{s0}+3\Delta$. Therefore, to ensure that $I \supseteq I_{R1}$ and $I \supseteq I_{P1}$, the sender uses I which contains $[c_s(t_{s0})-3\Delta,\ c_s(t_{s0})+13\Delta]$.

At t_{r0}, the Recipient confirms that the Post will accept m, σ_m, σ_r at t_{p2}, if the receiver sends them to the Post at t_{r1}, due to timeout while waiting for the affidavit of origin σ from the Sender. The timeout is after 2Δ; another Δ may be added due to the wakeup delay, hence $t_{r0}+2\Delta \le t_{r1} \le t_{r0}+3\Delta$. Also, $t_{p2} \le t_{r1}+\Delta$.
The Post bounds t_{r2} by $c_p(t_{p2})-\Delta<t_{p2}<t_{r2}<t_{p2}+\Delta<c_p(t_{p2})+2\Delta$. To make sure this validation will pass, the Recipient notice that the maximal bias between clocks is 2Δ, and therefore in particular $c_r(t_{r0})<c_r(t_{p2})-2\Delta<c_p(t_{p2})<c_r(t_{p2})+2\Delta<c_r(t_{r0})+6\Delta$. Hence, to ensure the validation by Post at t_{p2} passes, the Recipient should ensure that I contains $\boldsymbol{I_{R0} \equiv [c_r(t_{r0})-\Delta, c_r(t_{r0})+8\Delta]}$. Notice that this also ensures to the Recipient (at t_{r0}) that $t_{r0F}, t_{r2} \in I$. To ensure this will hold, the sender users I which contains $[c_s(t_{s0})-3\Delta,\ c_s(t_{s0})+11\Delta]$. We conclude that $\boldsymbol{I=[c_s(t_{s0})-3\Delta,\ c_s(t_{s0})+13\Delta]}$.

2.4.2. Validation of Affidavits in OTDP

The validation of affidavits in OTDP is almost identical to that in STDP. The only difference is that we allow the "R" affidavit to be based only on the signature[8] of the Sender, and allow either Sender or Post to sign the "A" affidavit, as follows.

$$V_{OTDP}("MR", a, m, response, I, \sigma_m, \sigma_{mr}) =$$

$$\begin{cases} True\ if\ Verify_{a.s.v}(<"M", a, m, I>, \sigma_m) \wedge Verify_{a.p.v}(<"MR", a, m, response, I>, \sigma_{mr}) \\ True\ if\ Verify_{a.s.v}(<"MR", a, m, response, I>, \sigma_{mr}) \\ False\ otherwise \end{cases}$$

$$V_{OTDP}("R", a, m, response, I, \sigma) = \begin{cases} True\ if\ Verify_{a.r.v}(<"R", a, m, response, I>, \sigma) \\ False\ otherwise \end{cases}$$

$$V_{OTDP}("F", a, m, I, \sigma) = \begin{cases} True\ if\ Verify_{a.p.v}(<"F", a, m, I>, \sigma) \\ False\ otherwise \end{cases}$$

$$V_{OTDP}("A", a, I, \sigma) = \begin{cases} True\ if\ Verify_{a.p.v}(<"A", a, I>, \sigma) \vee Verify_{a.s.v}(<"A", a, I>, \sigma \\ False\ otherwise \end{cases}$$

3. Model and Specifications

We consider a set *n* of processors (machines) running the Trusted Delivery Layer (TDL) protocol, which we call TDL machines (defined below); without loss of generality assume each machine has a unique identifier in the set *{1,…n}*. Each machine TDL has three *connectors,* as shown in **Figure 5**:

- Clock input connector *clock*, which provides real-time reading (with bounded offset) periodically.
- Transport Layer connector *T*, allowing TDL to communicate with other machines.
- Application Layer connector *A*, whereby the TDL layer interacts with the application.

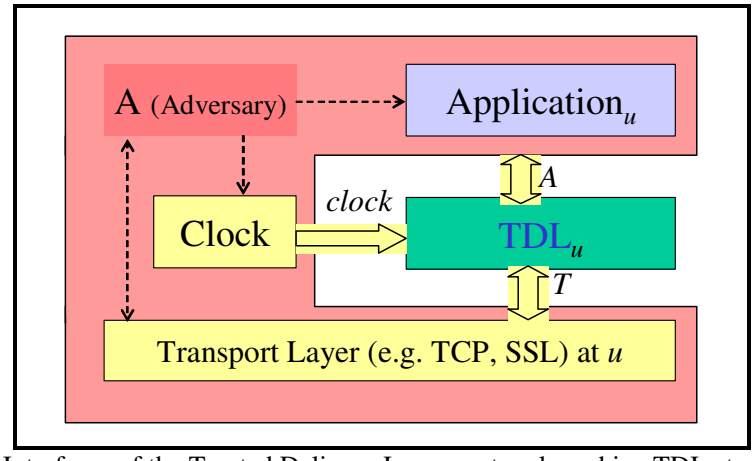

Figure 5: Interfaces of the Trusted Delivery Layer protocol machine TDL at processor *u*

[8] In many applications, the response is (often) known to the Sender (e.g. acknowledgement), so the Sender can sign only *once* by including in the signature σ_m the value $f(x)$ where f is a one-way function and $x \in_R \{0,1\}^n$ (a sufficiently-long random string), and exposing x to turn σ_m into σ_{mr}. The Post can use same method to save signatures. We expect these `tricks` to be done in real deployment, but ignored them for simplicity of exposition.

Since our goal is to provide an underlying foundation layer for secure e-commerce protocols and applications, it is critical that we provide well defined interfaces, specifications and proofs of security. However, modeling and analysis of cryptographic protocols is challenging. In particular, computational security is based on computational limitations of the adversary and provides only probabilistic security guarantees. This requires rigorous modeling and analysis, bounding the probability of a `bad` execution of the protocol, which is further complicated since, for worst-case analysis, we want to allow the adversary to control delays and clock biases (up to predefined limits).

In the first subsection below, we develop appropriate model with well-defined probability distribution of the possible behaviors (runs) of the system. The model allows us to specify security properties provided and assumed by each protocol, including integrity and confidentiality properties. To make our results more practical, we adopt the `concrete security` approach of [BKR94,BCK96], namely we bound the failure probability as a function of the block (and key) sizes and the number of queries/steps.

In the second subsection, we present the specifications for the Trusted Delivery Layer, consisting of prerequisites (required lower-layer services and allowed usage for upper layer), integrity specifications and confidentiality specifications.

3.1. Machines, Runs and Specifications

We now define a simple `adversarial model` where the adversary controls the entire environment of the protocol, including both infrastructure (clocks, communication) and application. This allows us to provide precise analysis of the probability distributions of runs and bound the failure probability as in [PW00, PWS00], but with simpler model and analysis, and adding support for concrete security (not merely asymptotic) and realistic timing and synchronization assumptions (not pure synchronous or asynchronous).

We first define a simple model for interacting, *cryptographic machines*, extending the I/O automata model of [L96], mostly by allowing (explicit) random input to the transition function. As in [L96] and most follow-up works, our definition of a machine assumes a `single threaded` processing model, i.e. does not consider events occurring while the system is handling another event. We believe this is an acceptable simplification, since the protocols we present require minimal computation, and since we allow the adversary to control all interactions between the machines.

To simplify and w.l.o.g., there is always a single input word x of length w, identified by a connector $c \in C$ where C is finite set of *connectors* (interfaces); e.g. for the TDL machine in **Figure 5** the connectors are $TDL.C=\{T, Clock, A\}$. The input to the transition function of machine M also contains the current state $s \in M.S[w,k]$, and a random-bit string r of length $M.r[w,k]$; both $M.S[w,k]$ and $M.r[w,k]$ depend on the word length w and the *security parameter* k. The output of the transition function consists of the new state s, identification of the output channel c, and three words, denoted O (which is the output to be placed on channel c), *id* and *otime*. The two last outputs, *id* and *otime,* are used only by the adversary: *otime* is the time till the next event *id* is the machine invoked in the next event (protocol machines always output *0* on both *id* and *otime*). As part of the specification of the machine, we assume a function $M.t$ mapping $\langle w, k \rangle$ to the time complexity (under an appropriate, specified computational model).

Definition 3-1 A cryptographic machine M is a tuple $\langle M.C, M.S, M.Init, M.\delta, M.t, M.r \rangle$ where for every integers w and k:

- $M.C=\{c_1,c_2,\ldots\}$ is a finite ordered set of identifiers (called *connectors*)
- $M.S[w,k]$ is a finite set (of states).
- $M.Init[w,k] \in M.S[w,k]$ (initial state).
- $M.\delta[w,k] : M.S[w,k] \times \{0,1\}^{M.r[w,k]} \times C \times \{0,1\}^{w} \rightarrow S \times C \times \{0,1\}^{w} \times \{0,1\}^{w} \times \{0,1\}^{w}$ is a (transition) function computable in time $M.t[w,k]$.

An <*w,k*> *input event* of machine *M* is a tuple *in*=<*"in",id, time, s, r, c, x*>, and an <*w,k*> *output event* is a tuple *out*=<*"out", id, time, s, c, O, oid, otime*>, where $r \in \{0,1\}^k$, $s \in M.S[w,k]$, $c \in C$ and *x, O, id, oid, otime* $\in \{0,1\}^w$ and *time* $\in \{0,1\}^*$. When there is no danger of confusion, we often omit the indexes *[w,k]*, and refer to an <*w,k*> *input (output) event* simply as an *input (respectively output) event*. We use dot notation to refer to specific elements of events, e.g. $v.s \in M.S[w,k]$ is the third element of event *v,* with $v.type \in \{$*"in","out"*$\}$ indicating the event type.

Let *in (out)* be an input (respectively output) event of machine *M.* We say that *out M-follows from in,* denoted $in \xrightarrow{M} out$, if *in.type="in"* and *out*=<*"out", in.id, in.time, M.*δ*[w,k](in.s, in.r, in.c, in.x)*>.

We consider configurations with one adversary machine *A* and $n<2^w$ *processors*, i.e. instances of the `honest` protocol machine *P.* The adversary invokes the processors and provides inputs on one of their connectors (but the adversary cannot directly control the state and random inputs). After a processor makes a transition, control always passes back to the adversary *A;* formally, we require that the identity output *oid* and the time output *time* of the protocol machine *P* are both always the fixed value 0^k, i.e. for every *w, k, s, r, c,* and *x* holds $P.\delta[w,k](s,r,c,x).oid, .otime=0^k$. We call such machines *protocol machines.*

The adversary machine *A* selects the processor *p* to be invoked by putting its identity in the *A.oid* output, and outputs a word and connector for the input to the processor, as well as a time value in the *A.otime* output. Once the processor made its transition based on the input word and connector provided by the adversary, the adversary receives the outputs and proceeds. We also allow the adversary machine to invoke itself rather than a processor, by putting in *A.oid* the value 0^k. Notice that we do not allow the adversary to corrupt processors adaptively as in [CFGN96], and certainly not to switch processors between corrupted and correct states as in proactive security [CGHN97].

Definition 3-2 Given (adversary) machine *A* and protocol machine *P* s.t. $A.C \supseteq P.C$, integers *w, k* and a sequence of (random) strings $R=\{r_i \in \{0,1\}^{max\{A.r(w,k),P.r(w,k)\}}\}$, we define an <*w,k,R*>*-run* of <*A,P*> as a sequence of <*w,k*> input and output events $\{in_1, out_1, in_2, out_2, ...\}$ such that $in_1.time, in_1.id=0^w$, $in_1.time=0^w$, $in_1.s=A.Init$, $in_1.r=r_1$, $in_1.c=A.C[1]$ and $in_1.x=0^w$ and for every $i>1$:

- $in_i.type=$*"in"*, $out_i.type=$*"out"*, i.e. in_i is an input event and out_i is an output event.
- If $in_i.id=0^k$ then out_i *A*-follows from in_i, otherwise out_i *P*-follows from in_i.
- $in_i.id=out_{i-1}.oid$, $in_i.c=out_{i-1}.c$, $in_i.x=out_{i-1}.x$, $in_i.time=out_{i-1}.time+out_{i-1}.otime$.
- $in_i.r=r_i$.
- Let *pre(i)* be the largest positive integer s.t. $pre(i)<i$ and $out_{pre(i)}.id=in_i.id$. If such *pre(i)* exists, then $in_i.s=out_{pre(i)}.s$; otherwise, $in_i.s=P.Init$.

For any given probability distribution of sequences of (random) strings *R,* e.g. the uniform distribution $U_l=(\{r_i \in_R \{0,1\}^l\})^*$ where $l= max\{A.r(w,k),P.r(w,k)\}$, we have a well-defined probability distribution of runs denoted <*w,k,R*>*-runs.* We usually consider only finite runs of *q* events; we denote their probability distribution <*w,k,R ,q*>*-runs of* <*A,P*>. When *R* is irrelevant or (as often) $R=U_{l,q}=(\{r_i \in_R \{0,1\}^l\})^q$ we simply refer to a <*w,k*>*-run* or <*w,k,q*>*-run* of <*A,P*>.

Our goal is to present *specifications*, defining functionality required from the protocol (in our case, from the trusted delivery layer), and then to present a specific protocol and prove that it fulfills the specifications. A specification is an ordered pair *Pre*➔*Guarantee* of (sets of) properties of runs, such that if *Pre* holds then the protocol should ensure that *Guarantee* holds. Since we consider realistic adversarial (cryptographic) setting, there is always some probability of the attacker to succeed in creating a bad run (e.g. when the attacker is lucky in guessing a secret key). We therefore define probabilistic preservation of specifications. We define a concrete security definition, and, based on it, also polynomial (asymptotic) definition.

Definition 3-3 A *property* is a predicate over runs. A *spec* (specifications) is an ordered pair <*prerequisites, guarantees*> of two sets of properties. A spec *pre*➔*guar* is *satisfied* with probability ε for

<w,k,q>-runs of <A,P>, if *Prob(pre(r)➔guar(z))>ε* for $z \in_R$ *<w,k,q>-runs of <A,P>*. Cryptographic machine *P is (t,rand,q,w,k,ε)-secure for spec pre➔ guar* if *P.t(w,k)≤t*, *P.r(w,k)≤rand* and spec *pre➔guar* is satisfied with probability *ε* for $<w,k,U_{r,q},q>$*-runs of <A,P>*, for every cryptographic machine *A* such that *A.C⊇P.C, A.t(w,k)≤t* and *A.r(w,k)≤r.* We say that *P* is *polynomially-secure for spec pre➔ guar* if it is *(t(k),r(k),q(k),w(k),k,ε(k))-secure for spec pre➔ guar*, for some polynomials *t, r, q, w* and *ε* (s.t. *ε(k)∈(0,1]*), and for sufficiently large *k*.

3.2. *Δ*-Sync Connectivity and Clock Specifications

Notation: If *e.id=p* then we say that *e* is an event of *p.* The value of variable *x* during event *e* of processor *p* in run *z* is denoted as $x_p@z.e$. When the run is obvious or irrelevant we omit it and write simply $x_p@e$. If *e.C="clock" (e.C="A", e.C="T")* then we say that *e* is a clock (respectively, application, transport) event. Let *TIME* be a function mapping the output binary strings on port *time* to positive real numbers. The **real-time** of an event *e* in run *z*, denoted *time(e),* is *TIME(e.t).*

We require the clock events to satisfy three bounds: at most *Δ/ρ* from one clock event to the next; at most *Δ/2* offset between the value of the clock in any clock event and the real time; and at most *Δ/4* change in the offset between two clock readings in the same processor. All requirements are in line with the practice and theory of clock synchronization protocols, in benign and adversarial settings, see e.g. [M90,L96]. The last requirement is less common in analytical models, and indeed it could be removed, but this seems to result in significant increase in the uncertainty of timeout events (we omit the details for lack of space).

Definition 3-4 Let *z* be a *<w,k,q>*-run of *<A,P>*. We say that processor *p* is *(Δ,ρ)*-synchronized during *z*, or that $sync_{p,[\Delta,\rho]}(z)=True$, if:

(a) [there is a clock event at least once every *Δ/ρ*] for every interval *[t,t+Δ/ρ]⊆[time(z[0]), time(z[q])],* there is an event *clk∈r* s.t. *clk.id=p*, *clk.C="clock", time(clk)∈[t,t+Δ/ρ],* and

(b) [the offset of every clock input from real-time is at most *Δ/2*] for every event *clk∈z* s.t. *clk.id=p*, *clk.C="clock"* and *clk.type="in"* holds *|time(clk)-TIME(clk.x)|≤Δ/2*, and

(c) [the offset from real-time of the same clock changes by at most *Δ/4*] for every pair of events *c[0], c[1]∈z* s.t. for *i∈{0,1}* holds *c[i].id=p*, *c[i].C="clock"* and *c[i].type="in"* holds *|*(TIME(*c[0].x)-time(c[0]))-(TIME(c[1].x)-time(c[1]))|≤Δ/4*,

We next bound the message transmission delay and assume network addresses *addr* are given.

Definition 3-5 Let *z* be a *<w,k,q>*-run of *<A,P>*. We say that *u∈{1,…n}* is *Δ-connected* to *v∈{1,…n}* during *z* if every event $e_u \in z$ s.t. e_u*.C="T", *e_u*.id=u, *e_u*.x=(m,addr[v]), *e_u*.type="out",* there is a corresponding event $e_v \in z$ s.t. e_v*.id=v, *e_v*.x=(m,addr[u]), *e_v*.C="T", *e_v*.type="in"* such that *time(*e_v*)<time(*e_u*)+Δ*. Also, for *y∈{u,v}* there is event $e_I \in z$ s.t. e_I*..id=z, *e_I*..x=("addr",addr[y]), *e_I*.C="T", *e_I*.type="in", time(*e_I*)<time(*e_y*).* We say that *u,v are Δ-connected* if *u* is *Δ-connected* to *v* and vice verse. Predicate $\boldsymbol{Sync\&Conn_{V,[\Delta,\rho]}(z)}$ is true for *set V* of processors, if for any *u,v∈V* holds $sync_{p,[\Delta,\rho]}(z)=True$ and *u,v are Δ-connected during z.*

This properties map to the more natural and easier to use properties of Definition 2-1.
Lemma 3-1 For any run *z*, if $Sync\&Conn_{\{u,v\},[\Delta,\rho]}(z)$ then *u,v* are *[Δ,ρ]-Sync connected* in *z.*■

The protocol will send addresses and time intervals in messages, and therefore they must not be too long, so that the protocol can send, using the T channel, messages of total length at most *w*. Specifically we require that clock values and addresses are of length at most *w'<<w* (i.e. all other bits are zero). Namely:

Definition 3-6 Given *<w,k>*-run *z* and integer *w'<w*, let $suffix_{w'}[z]=Ok$ if for all *v∈{1,…n}* holds *|addr(v)|≤w'* and for every event *clk∈z* s.t. *clk.C="clock"* and *clk.type="in"* holds *|clk.x|≤w'*.

3.3. Usage Specifications

We now define the correct usage of the TDL, i.e. the allowed interactions on the A connection.

Definition 3-7 Let z be a $<w,k,R,q>$-run of $<A,P>$, and $s,r,p \in \{1,\ldots n\}$. Then $TDL\text{-}use_{s,r,p,w'}(z)=Ok$ *if:*

(a) There is exactly one $init_u$ input event for $u \in \{s,r,p\}$, with inputs k, w' (s.t. $w'<<w$). Denote the corresponding output events as i_s , i_r and i_p respectively, with output $x.v$ (verification key) and $x.addr$ (network/IP address).
(b) There is a single input event a_u , for $u \in \{s,r,p\}$, s.t. $a_u.C="A"$, $a_u.type="in"$, and $a_u.x=("Agreement",a)$. Furthermore, for $y \in \{s,r,p\}$, $a.y.v=i_y.x.v$, $a.y.addr= i_y.x.addr$.
(c) There is a single input event in to s with $in.C="A"$, $in.x=("deliver",m)$, $|m|<w'$. There is no "deliver" event in r, p. We denote this event $deliver(z)$.
(d) There is at most one input event in to r with $in.C="A"$, $in.x=("response",r)$ and no such event in s,p. We denote this event $response(z)$.
(e) There are no additional input event in with $in.C="A"$.

3.4. Analysis of OTDP

The security of the protocol is derived from the security of the signature scheme used, as defined[9] in [BR96,GB01]. We first note a simple bound on the length of the messages, assuming we use signature scheme S with running time (for key generation, signature and verification[10]) up to τ, producing signatures and keys of length at most[11] w' and using at most w' random bits, for inputs of length up to w and security parameter k; we say that $bounds(\tau,w',w,k)[S]=Ok$.

Lemma 3-2 Let S be a signature scheme s.t. $bounds(\tau,w',w,k)[S]=Ok$ with $w \geq 7w'$. Then for any run z of OTDP(S) s.t. $TDL\text{-}use_{s,r,p,w'}[z]=Ok$ and $suffix_{w'}[z]=Ok$, holds:
(a) The length of the agreement a is at most $6w'$.
(b) The length of the input to each affidavit is at most $9w'$.
(c) The length of the longest message placed by OTDP on the T interface (i.e. sent via the transport layer) is $7w'$.
(d) The protocol uses at most w' random bits at each event.

Proof: Immediate, by counting the length of inputs and from the definitions above. ■

We next obtain our main result:
Theorem 1 For any integers $w', rand, q, t, \tau, k$ and any $\varepsilon \in (0,1)$, let S be a $(t \cdot q,q,\varepsilon)$-secure (against adaptive Chosen Message Attack) signature scheme s.t. $bounds(\tau,w',7w',k)[S]=Ok$, $t>3\tau$ and $rand \geq w'$. Then OTDP(S) is $(t,rand,q,7w',k,\varepsilon)$-*secure for* $(TDL\text{-}use_{s,r,p,w'} \wedge Sync\&Conn_{\{s,,r,p\},[\Delta,\rho]} \wedge suffix_{w'}) \rightarrow TDL_{s,r,p,[\Delta,16\Delta]}$.
Proof: follows by standard reduction argument; details omitted and will be given in final version. ■

The asymptotic-security version of the result is, of course, simpler – and follows easily.
Corollary: Let S be a polynomially-CMA-secure signature scheme. Then OTDP(S) is polynomially-secure for $(TDL\text{-}use_{s,r,p} \wedge Sync\&Conn_{\{s,,r,p\},[\Delta,\rho]}) \rightarrow TDL_{s,r,p,[\Delta,16\Delta]}$.

4. Possible Extensions and Deployment Issues

We now discuss some possible extensions and improvements to the protocols, and other deployment issues.

4.1. Confidentiality of message from Post

To protect the confidentiality of the message, the application at the sender should encrypt it using the key of the receiver, before delivering it via the TDL layer (OTDP). To preserve the notarization (non-repudiation) properties and prevent key-spoofing attacks [AN95,AN96], the Post should also sign the public encryption

[9] We omit the limit on computations of the hash allowed to the adversary, which is irrelevant to our results.
[10] One can easily express the substantial differences in time complexities of sign, verify and key-generate.
[11] Signature schemes usually produce short, fixed length output and require a fixed number of random bits. Notice a minor change is required here to use signature schemes with message recovery.

key . Encryption should use a committing-encryption scheme, as defined in [GH03]; this could be a public key committing encryption scheme (but we could also use a shared-key scheme in which case we sign the public commitment key instead of signing the encryption key). Specifically, we could use any public key cryptosystem where the randomness used (e.g. for padding) during encryption is recovered during decryption, such as (padded) RSA (this is also argued in [ADR02]. The validation process will require, as additional input, the randomness used for encryption, and then will simply repeat the encryption process to confirm that the ciphertext corresponds to the given plaintext.

We comment that most time-stamping proposals, e.g. [HS91], protect confidentiality of the document from the notary by time-stamping the cryptographic hash of the message. This is a natural heuristics, and secure under `random oracle` analysis, but not using standard definitions of cryptographic hash functions (e.g. one-way and collision resistant hash functions). Therefore, there is an advantage to using the technique above, even if only a timestamp is needed (no receiver involved).

Notice that some applications where confidentiality is a concern, e.g. bidding, may require also *non-malleability* [DDM00], i.e. prevent eavesdropper (or corrupted Post) from sending bids which depend on the (unknown) value in an encrypted and/or committed bid. In this case, use non-malleable encryption and/or commitment schemes; see [DDM00].

4.2. Integrity and Availability: Distributed Post Office

The protocols presented extend naturally to support multiple, distributed Post offices. A trivial solution is to run multiple executions of the protocol with multiple Post offices, and define an affidavit to be valid if and only if signed by a sufficient number of Post offices. Or, the post offices could use shares of the private key of a threshold (distributed) signature scheme, in which case validation of affidavits would remain as in the existing protocol. The necessary changes to the protocol are obvious.

4.3. Implementation Comments

Server / Stateless implementation. The protocol as described instructs the Post, in several cases, to send a message to either sender or receiver (not as a response), thereby deviating from the classical client-server paradigm. It also requires the post to maintain state (e.g. to detect timeouts). There are advantages to using a purely client-server design (i.e. the Post only sends responses, never initiates communication), as well as to use a stateless Post. It seems fairly easy to modify the protocol (at some cost in parameters) to either a purely client-server design or to a stateless Post design; we omit the details and analysis. However, we don't see how a trusted delivery protocol could use a stateless, purely client-server design.

No signatures by Sender and Recipient. In some scenarios, the receiver and sender do not have certified public keys, or cannot perform public key signatures; see such a protocol [AGHP02]. The STDP can be modified to support this scenario.

5. Conclusions and Further Research

Trusted delivery services are required in many commercial scenarios, e.g. for business-to-business commerce and for banking and payment transactions. We present a practical and provably secure protocol, which is the first to provide timestamps (with bounded uncertainty) and bounded termination, under realistic, semi-synchronous assumptions. This protocol could provide a foundation layer over which more advanced secure e-commerce services may be build. Our model of adversarial environments could also be useful for other protocols and tasks, esp. related to e-commerce.

We hope our real-time adversarial model may be useful to analyze lower layer security and fault-tolerance protocols (e.g. TCP, TLS), and higher layer secure e-commerce protocols; we also hope it will be possible to prove general composition theorems as in [PSW00]. Finally, real systems are multi-threaded (even in a single processor), motivating appropriate extensions to our (single-threaded) model.

References

[AA*93] Y. Afek, H. Attiya, A. D. Fekete, M. Fischer, N. Lynch, Y. Mansour, D. Wang, and L. Zuck. Reliable communication over unreliable channels. Journal of the ACM, 41(6):1267--1297, 1994.

[ADR02] Jee Hea An, Yevgeniy Dodis and Tal Rabin, On the Security of Joint Signature and Encryption, in Theory and Application of Cryptographic Techniques, pp. 83-107, 2002.

[AGHP02] Martín Abadi, Neal Glew, Bill Horne and Benny Pinkas, Certified Email with a Light On-Line Trusted Third Party: Design and Implementation, Proceedings of the Eleventh International World Wide Web Conference (May 2002), 387-395.

[AL93] Abadi, M. and L. Lamport, "*Composing Specifications*," ACM Transactions on Programming Languages and Systems, Vol. 15, No. 1, January 1993, pp. 73-132.

[AN95] Ross Anderson and Roger Needham. Robustness principles for public key protocols. In Proc. Int'l. Conference on Advances in Cryptology (CRYPTO 95), volume 963 of Lecture Notes in Computer Science, pages 236--247. Springer-Verlag, 1995. hPost://citeseer.nj.nec.com/article/anderson95robustness.html

[AN96] Abadi, M. and Needham, R. 1996. Prudent engineering practice for cryptographic protocols. IEEE Trans. Softw. Eng. 22, 1 (Jan.), 6-15. hPost://citeseer.nj.nec.com/abadi96prudent.html

[ASW97] N. Asokan, M. Schunter, and M. Waidner. Optimistic Protocols for Fair Exchange. In Proceedings of 4th ACM Conference on Computer and Communications Security, Zurich, April 1997.

[ASW00] N. Asokan, V. Shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures. IEEE J. Selected Areas in Comm., 18(4):593-610, April 2000.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk, Pseudorandom functions revisited: The cascade construction and its concrete security. Proceedings 37th Annual Symposium on the Foundations of Computer Science, IEEE, 1996.

[BKR94] Mihir Bellare, Joe Kilian and Phillip Rogaway. *The security of the cipher block chaining message authentication code.* Journal of Computer and System Sciences, Vol. 61, No. 3, Dec 2000, pp. 362--399. Extended abstract in Advances in Cryptology - Crypto 94 Proceedings, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed, Springer-Verlag, 1994.

[BR94] Mihir Bellare and Phillip Rogaway. *Optimal Asymmetric Encryption*. In Alfredo De Santis, editor, Advances in Cryptology - EUROCRYPT '94, Perugia, Italy, May 9-12, 1994, volume 950 of Lecture Notes in Computer Science, pages 92-111. Published by Springer-Verlag, 1995.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. *Adaptively secure multi-party computation*. In Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC '96), pages 639--648. ACM, 1996.

[CGHN97] Ran Canetti, Rosario Gennaro, Amir Herzberg, and Dalit Naor. *Proactive security: Long-term protection against break-ins*. RSA Laboratories' CryptoBytes, 3(1), 1997.

[DDN00] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. SIAM Journal on Computing, 30(2):391--437, 2000.

[EGL85] Shimon Even, Oded Goldreich, Abraham Lempel: A Randomized Protocol for Signing Contracts; Communications of the ACM 28/6 (1985) 637--647.

[G82] Oded Goldreich. A protocol for sending certified mail. Technical report, Computer Science Department, Technion, Haifa, Israel, 1982.

[GB01] Shafi Goldwasser and Mihir Bellare, Lecture Notes on Cryptography, Available online at hPost://www.cs.ucsd.edu/users/mihir/papers/gb.html, 1996-2001.

[GH03] Yitchak Gertner and Amir Herzberg, Commited Encryption, work in progress, 2003.

[GJM99] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In Advances in Cryptology: Proceedings of Crypto'99, volume 1666 of Lecture Notes in Computer Science, pages 449--466. Springer-Verlag, 1999.

[HPS01] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In Proc. of The ACM Conference on Electronic Commerce (EC'01), Tampa, Florida, October 2001.

[HS91] Stuart Haber and W.-Scott Stornetta. How to Time-Stamp a Digital Document. Journal of Cryptology, 3(2):99--111, 1991. hPost://citeseer.nj.nec.com/haber91how.html

[ISO 7498-2] ISO 7498-2: 1989, *Information processing systems – Open systems interconnection – Basic reference model – Part 2: Security Architecture*.

[ISO 10021-2] ISO/IEC 10021-2: 1988, *Information Technology – Text Communication – Message-oriented text interchange systems (MOTIS): Overall architecture*.

[ISO13888-1] ISO/IEC 3rd CD 13888-1. Information technology – security techniques – Non-repudiation, Part 1: General model. ISO/IEC JTC1/SC27 N1274, March 1996.

[ISO13888-3] ISO/IEC 2nd CD 13888-3. Information technology – security techniques – Non-repudiation, Part 3: Using asymmetric techniques. ISO/IEC JTC1/SC27 N1379, June 1996.

[J98] Mike Just. Some Timestamping Protocol Failures. In Internet Society Symposium on Network and Distributed System Security, 1998. Available at hPost://citeseer.nj.nec.com/just98some.html.

[KMZ02] An Intensive Survey of Non-repudiation Protocols. Steve Kremer, Olivier Markowitch & Jianying Zhou . To appear in Computer Communications Journal. Elsevier. 2002.

[L96] Nancy Lynch, Distributed Algorithms, Morgan Kaufman, San Francisco, 1996.

[LPSW00] Gerard Lacoste, Birgit Pfitzmann, Michael Steiner, Michael Waidner (Editors), SEMPER - secure electronic marketplace for Europe, Vol. 1854, Springer-Verlag, ISBN = "3-540-67825-5".

[M90] Mills, D.L. *On the accuracy and stability of clocks synchronized by the Network Time Protocol in the Internet system*. ACM Computer Communication Review 20, 1 (January 1990), 65-75.

[M97] Silvio Micali, Certified E-Mail with Invisible Post Offices - or - A Low-Cost, Low-Congestion, and Low-Liability Certified E-Mail System; presented at 1997 RSA Security Conference, San Francisco.

[M03] Silvio Micali, Simple and Fast Optimistic Protocols for Fair Electronic Exchange, Proc. 22nd Symp. on Principles of Distributed Computing (PODC), Boston, Massachusetts, July 2003, pp. 12-19.

[MSP96] National Security Agency. Secure Data Network System : Message Security Protocol (MSP), January 1996.

[PC*03] Jung Min Park, Edwin Chong, Howard Siegel, Indrajit Ray, Constructing Fair-Exchange Protocols for E-commerce Via Distributed Computation of RSA Signatures, to be published in PODC'03, Boston, July 2003.

[PSW00] Birgit Pfitzmann, Matthias Schunter, Michael Waidner, Provably Secure Certified Mail, IBM Research Report RZ 3207 (#93253), IBM Research Division, Zurich, Feb. 2000.

[R00] Eric Rescorla. SSL and TLS: Designing and Building Secure Systems. Addison-Wesley, 2000.

[R98] Tal Rabin. A Simplified Approach to Threshold and Proactive RSA. In H. Krawczyk, editor, Advances in Cryptology--CRYPTO'98, Lecture Notes in Computer Science Vol. 1462, pp. 89--104, Springer-Verlag, 1998.

[R99] Meelis Roos, Integrating Time-Stamping and Notarization, Master Thesis, Tartu University, Faculty of Mathematics, May 1999.

[RFC2246] T. Dierks, C. Allen, The TLS Protocol: Version 1.0, Network Working Group, Internet Engineering Task Force (IETF). Available online at hPost://www.ietf.org/rfc/rfc2246.txt.

[RFC2411] R. Thayer, N. Doraswamy and R. Glenn, IP Security Document Roadmap, Network Working Group, Internet Engineering Task Force (IETF). Available online at hPost://www.ietf.org/rfc/rfc2411.txt. November 1998.

[RFC2560] Michael Myers, R. Ankney, A. Malpani, S. Galperin, and Carlisle Adams. RFC2560: X.509 Internet Public Key Infrastructure Online Certicate Status Protocol - OCSP, June 1999, hPost://www.ietf.org/rfc/rfc2560.txt.

[RFC3161] C. Adams, P. Cain, D. Pinkas, R. Zuccherato, RFC 3161: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), IETF Network working group, August 2001, hPost://www.ietf.org/rfc/rfc3161.txt.

[Sy96] P. Syverson. Limitations on Design Principles for Public Key Protocols. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages 62--73, Oakland, CA, May 1996. hPost://citeseer.nj.nec.com/syverson96limitation.html

[X.402] ITU-T Recommendation X.402 (11/95), *Information Technology – Message Handling Systems (MHS): Overall Architecture*.

[X.800] ITU-T Recommendation X.800 (1991), *Security Architecture for Open Systems Interconnection for CCITT Applications*.

[Z01] Jianying Zhou. "Non-repudiation in Electronic Commerce". Computer Security Series, Artech House, August 2001

[ZG96] Jianying Zhou and Dieter Gollmann. Observations on non-repudiation. In Kim Kwangjo and Matsumoto Tsutomu, editors, Advances in Cryptology - Asiacrypt 96, vol. 1163 of LNCS, pp. 133--144. Springer-Verlag, 1996.

[ZG97] Jianying Zhou, Dieter Gollmann. *Evidence and non-repudiation*. Journal of Network and Computer Applications, London: Academic Press, 1997. hPost://citeseer.nj.nec.com/zhou97evidence.html