Guaranteed Delivery for Secure Electronic Commerce and Payments

Draft version: Wednesday, June 26, 2002

-- Comments Welcome!! --

Amir Herzberg, amir@herzberg.name
Computer Science Dept.
Bar-Ilan University

Abstract

We present the Guaranteed Delivery Protocol for ensuring non-repudiation of submission and of origin, for time-sensitive messages. The protocol allows third parties (Delivery Authorities or notaries) to provide signed, timestamped receipts to the origin and destination of the message. The Delivery Authority (DA) may be distributed, for fault tolerance and proactive security. The message may be confidential, even from the DA; to achieve this we introduce the Commit then Encrypt then Sign *(CtEtS) method, allowing a trusted third party to sign a statement concerning a document without exposure to its contents; this method may have additional applications (e.g. timestamping, certified delivery). Guaranteed delivery is an important enabler for secure e-commerce applications. We demonstrate applications to secure payments, banking, bidding, auctions, gambling and business-to-business transactions such as supply chain.*

Keywords: secure electronic commerce; non-repudiation; timestamp; certified delivery; guaranteed delivery; secure messaging; delivery authority; secure payments.

1. Introduction

The Internet, with its global availability and support for automation, has a huge potential for improved services and commerce. However, security is often a concern. There are well-established standards for securing the communication among the parties. In particular, secure e-commerce applications often assume that communication among participants is authenticated and confidential. These security guarantees are provided by standard secure communication mechanisms that are part of the communication services. Specifically, e-commerce applications often assume that communication is secured in the Internet layer, using the IP-Security standard [RFC2411], in the transport layer, using TLS or SSL [R00,RFC2246], or in the application layer, e.g. using S/MIME [RFC2633,RFC2634]. See the network layers in **Figure 1** below.

However, for many e-commerce applications, there are additional security requirements beyond secure communication. These additional security requirements are often the result of the conflicting interests of the parties. It is possible to address such additional security requirements in each application individually. However, it is clearly desirable to identify common services required by many applications, and provide them by a common, underlying mechanisms.

In particular, many secure electronic commerce applications require some *non-repudiated message delivery services*. Such services are applied to messages or documents sent from one party (the *origin*) to another (the *destination*). Non-repudiated delivery services can simplify the design and analysis of higher-layer mechanisms and applications, which can now assume a stronger (and therefore easier) environment. This extends the traditional communication network layers (e.g. of ISO or TCP/IP) with an additional layer, the *non-repudiated delivery layer,* as a foundation for secure e-commerce applications.

Non-repudiation Services. Non-repudiation is a fundamental service required by many secure e-commerce applications. The most important non-repudiation services are:

1. **Non-Repudiation of Origin (NRO),** allowing the destination to prove that a message it delivered came from a specific, identified origin; the proof may also indicates the time of delivery.
2. **Non-Repudiation of Receipt (NRR),** allowing the origin to prove that the destination received the message, at particular time. Some authors require that this proof is signed by the destination, and define also non-repudiation of delivery (NRD) as a proof of message delivery signed by a third party (e.g. post office).
3. **Non-Repudiation of Submission (NRS),** allowing the origin to prove that it submitted the message, to the destination, at particular time. Namely, the origin has fulfilled its obligations for delivering the message; if it was not received this is due to a failure of the destination. The destination will normally receive the submitted message, except if there is a failure in the delivery of messages to it.

Non-repudiation of origin is usually provided by the origin digitally signing the message, together with the identity of the destination (and optionally the time). If the public key of the origin may be revoked, non-repudiation of origin is ensured by attaching (to the signature) proof that the origin's public key was valid at the time.

Non-repudiation of submission and of receipt is harder to obtain; the basic problem is that the destination may fail to receive or acknowledge the message, intentionally or otherwise; in which case there certainly cannot be a receipt (and therefore non-repudiation of receipt). Solutions require the parties to agree (in a non-repudiated manner, e.g. written contract) on a mutually trusted third party. Such a third party is called *notary, proxy,* or – and in this paper – *Delivery Authority (DA)*.

Once the parties agree on a delivery authority, non-repudiation for submission seems easy. The delivery authority provides the origin with *signed, timestamped submission receipt* for submissions sent via it. The origin can use this receipt in resolving disputes with the destination over damages due to the fact that the destination did not handle the message in time. This simple, `folklore` protocol has not received much attention, implementation or analysis (it was sometimes implied as a part of more complex non-repudiation protocols, e.g. in [ISO13888-3, ZG96]). In this manuscript, we focus on the *guaranteed delivery* service, which provides non-repudiation of origin and of submission, refining, analyzing and extending the `folklore` protocol. Most of the

existing works focus on *certified delivery*, which provides non-repudiation of origin and of receipt.

Certified delivery. Even an agreed upon delivery authority cannot produce proof of message receipt by the destination, in particular, when the destination is unable or unwilling to receive the message. Many works solve this problem by requiring the delivery service to be *atomic* (or *fair*): the message is delivered if and only if a signed receipt is provided (ensuring NRR). A *certified message delivery service* provides atomic/fair delivery with non-repudiation of origin and of receipt; corresponding to the postal services of `certified mail with receipt for delivery`. Variants of the certified delivery problem require the destination to send specific documents instead of or in addition to the receipt, e.g. *contract signing* and *fair exchange.* Several messaging systems and standards support certified delivery services, in particular [X.400,MSP96,SEMPER], and ISO standards [ISO13888-1,ISO13888-3]. Much of the research on certified delivery focuses on avoiding the use of a delivery authority, by probabilistic and/or gradual exchange, as in [G82, EGL85], or using it only to handle exceptions as in [ASW97,M97,PSW00,ZG96]. For overview and bibliography of certified delivery protocols, see [KMZ02,Z01].

For many applications, guaranteed delivery, ensuring non-repudiation of submission, is a more appropriate and useful service. Guaranteed delivery does not depend on the cooperation of the destination; the delivery authority provides the proof of submission to the origin. This corresponds to regular certified mail, which is usually the required mechanism for legally binding, non-repudiated delivery of messages in existing business and commerce relationships. The guaranteed delivery problem also appears easier than the certified delivery problem, leading to simpler and more efficient protocols. In this paper we focus on guaranteed delivery.

Organization and contributions. In this work, we suggest that a guaranteed delivery service, ensuring non-repudiation of submission and non-repudiation of origin, can be an important foundation layer for many secure e-commerce applications. We therefore propose that this (relatively simple) task should receive much more attention than it received so far, analyzed carefully, implemented and standardized. In this manuscript, we take the first steps in this direction, by presenting and motivating the guaranteed delivery problem, and presenting several protocols for guaranteed delivery.

In Section 2, we define the guaranteed delivery service; this definition critical to allow simplified design of applications taking advantage of this layer. The definition is not as trivial as one may expect (based on this being `folklore` mechanism for so long), especially when considering (bounded) delays and clock drifts, as required for realistic applications. Additional complications are due to the dependency on computationally secure cryptographic mechanisms (e.g. signatures), which may be broken with negligible probability or with overwhelming computational resources.

In Section 3, we present a simple guaranteed delivery protocol. The protocol handles bounded message delays and clock drifts. For simplicity, this protocol does not provide message confidentiality.

In Section 4, we extend the protocol to provide message confidentiality, even from the Delivery Authority (DA). This requires a new method for combining signature and

encryption operations, which we call *Commit then Encrypt then Sign (CtEtS);* this method is related to, but differs from, the recently proposed *Commit then Encrypt and Sign (CtE&S)* [ADY02] (their goal was to allow parallel computation of encryption and signature operations, while our motivation is to allow a third party, the DA, to provide non-repudiation for hidden information). The *CtEtS* method may have additional applications, e.g. for certified delivery, timestamping, and other tasks where a trusted third party is trusted for authenticating a document which is confidential (and not to be revealed even to the trusted third party).

The security of the applications depends on the security of the delivery service, as provided by the Delivery Authority. In Section 5, we sketch some results on security in the presence of DA faults (and recoveries). We first show, in Subsection 5.1, that by a minor extension, namely adding the signature of the origin on the delivery request, we can ensure non-repudiation of origin holds even if the DA may be corrupted. This requires the origin and destination to agree on the origin's public key in advance, without allowing revocation of this key.

We then extend, in Subsection 5.2, the protocols of the previous sections to use multiple, redundant Delivery Authorities. The resulting protocol can handle arbitrary (possibly malicious) failures of Delivery Authorities. Every DA may fail, as long as the total number of authorities corrupted within any bounded time interval is under a pre-defined threshold. We allow recovery of a corrupted DA, and do not assume detection of failures or recoveries, thereby achieving *proactive security,* building on the results in this area, see e.g. [BHHN00, CGHN97, CHH01,HJJ*97]. Precise definition and analysis of the security this distributed/proactive version of the protocols, providing guaranteed delivery with faults and recoveries, is beyond the scope of this paper.

Both certified delivery and guaranteed delivery services are a natural services for a non-repudiated delivery layer for secure e-commerce applications, as shown in **Figure 1**. The non-repudiated delivery layer can use secure communication (authentication and confidentiality) between the distributed authority and both origin and destination, provided by security facilities in the application layer (e.g. S/MIME [RFC2633], XML security [H02a]), transport layer (e.g. SSL / TLS [R00,RFC2246]) or Internet Layer (IP-Sec [RFC2411]). A non-repudiation delivery layer was proposed in SEMPER [LPSW00], but only with certified delivery service. Offering multiple non-repudiated delivery services resembles the offering of multiple services in the transport layer (UDP, TCP and others). In Section 6, we discuss several secure e-commerce applications built on top of the guaranteed delivery service, including secure payments, banking, brokerage, bidding, auctions, gambling and business-to-business transactions such as supply chain.

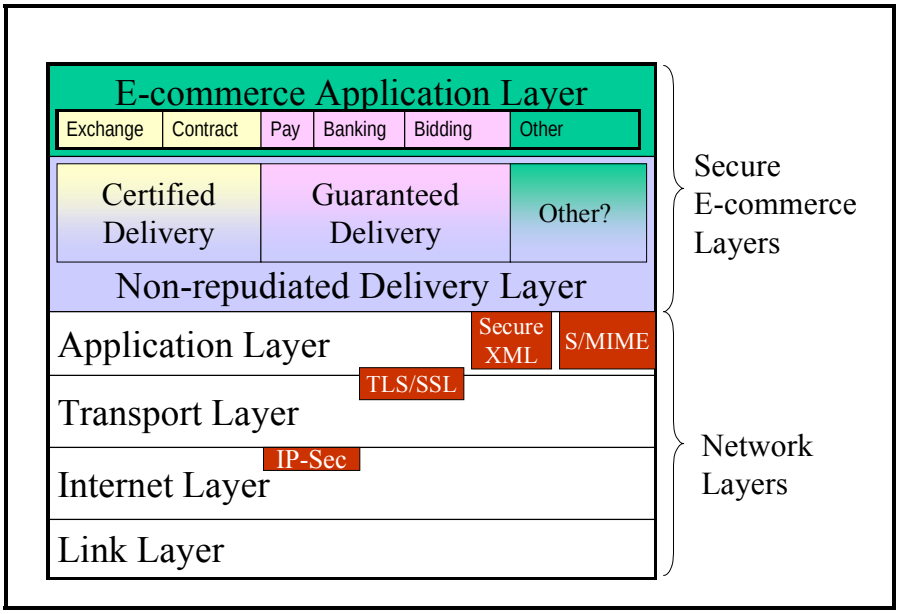

Figure 1: Secure E-Commerce and Network Layers

All of our protocols are efficient and practical. The simple protocol of section 3 does not require the origin to perform any public key operations, allowing implementations on computationally limited client platforms and with standard client software, and avoiding the well-known challenge of installing `local wallet software`. In Section 7, we discuss optimistic versions of the protocols, which have improved performance in the typical, no-faults case.

To help readability, Appendix A contains tables of symbols.

Non-repudiation and timestamping. The guaranteed delivery service provides the destination *non-repudiation of origin,* by delivering the document with a *timestamp.* The timestamp allows the destination to prove to a third party, e.g. judge, that the origin sent the document to the destination (at a particular time), and in particular, that the public key used to validate the origin was valid (not revoked) at the time of delivery. This deals with the risk that the destination would accept a document as properly signed by the origin by validating the origin's public key, but later the origin will claim that its public key was already revoked and therefore it is not responsible for the document.

Two (proposed) IETF standards already address the need for validating the public key of the origin of a message upon delivery. The Online Certificate Status Protocol (OCSP) [RFC2560] allows the destination (relying party) to check the validity of the public key by consulting an online Certification Authority (CA) or an Authorized Responder; however this does not provide proof that a particular (signed) message was received at that time, therefore OCSP is not sufficient for non-repudiation of origin. The Time Stamp Protocol (TSP) [RFC3161] provides time-stamps for arbitrary documents, and therefore can be used by the origin or destination to provide a timestamp for the (signed) message and the certificate, and thereby non-repudiation of origin. Indeed, we believe that our results can be applied as extensions to OCSP and TSP, and in particular that the services performed by the trusted third party in our protocols (the Delivery Authority, DA) are a natural extensions of the services of the trusted third party in OCSP (CA and/or Authorized Responder) and in TSP (the Time

Stamping Authority, TSA). The main additional service in our protocols compared to OCSP, TSP, and other timestamping solutions, is delivery of the document to the destination, thereby providing non-repudiation of submission (with timestamped proof of submission) as well as non-repudiation of origin (with timestamped document and proof of origin). A Delivery Authority combines the functions of a CA and Authorized Responder, as in OCSP [RFC2560], a Time Stamping Authority, as in TSP [RFC3161], and a mailbox server as in POP3 [RFC1939].

All the above methods, and also our protocols, rely on the delivery authority (or, in Section 5, on a sufficient number of authorities) to provide correct time-stamp. This implies that a penetration of the DA, or sufficient number of DA's in the future, will allow forgery. Some works address this problem, by `linking` the timestamps so as to prevent backdating even if all DA keys are exposed [R99]. We do not address this issue, although our protocols can be used with forward-secure signatures [BM99] for the same effect.

Timestamping Aspects. There are also some differences in the timestamping aspects of our solution compared to previous time-stamping solutions. We provide *absolute timestamps*, linking the delivery to a particular value of real time. Most previous results, e.g. [HS91,J98], focused on *relative timestamps,* creating linkage and thereby total or partial ordering among different timestamps. We do not see how to use relative timestamps for non-repudiation of delivery or of origin.

Absolute timestamping has a simple, `folklore` solution: send the hash of the document to the time-stamping server, and the timestamp is the server's signature on the time together with the hash of the document. One goal of signing a hash of the document (rather than the document directly) is to protect confidentiality of the document, even from the time-stamping authority (or delivery authority). Actually, to preserve confidentiality, the timestamp should be on a commitment to the document rather than on its hash; see details in Section 4. We also show, in Section 5, how to extend absolute timestamping to withstand time-stamping server failures (cf. [J98]).

2. Model and Requirements

We now present our model of the system and define the requirements from a guaranteed delivery service, as well as our assumptions about the underlying mechanisms (communication and clock synchronization). We tried to provide well defined model without making it overly complex and verbose; the interested reader should be able to add details and present our results using probabilistic I/O automata [L96], as done for the related problem of certified delivery in [PSW00].

We consider a network with a set P of *processors*. For simplicity, we focus on the delivery of a single message from processor $Org \in P$ *(origin)* to processor $Dest \in P$ *(destination).* (To handle multiple messages, use separate executions of the protocol, each with unique identifier.) A *protocol* P is a mapping, for each processor in P, of a tuple $(IE, OE, S, init, \delta)$, where IE (OE) is the set of input (respectively output) events; S is the set of states, with $init \in S$ the initial state; and δ is a probabilistic state-transition function from current state and input event, to next state and possibly one or more output events.

The events define interactions with the application (higher layer) and with the underlying communication and clock services, as illustrated in Figure 2. The only input from the application is a *deliver* event which defines a message $m \in \{0,1\}^*$ to be delivered. For simplicity, and without loss of generality, we assume that there is at most one deliver event in the execution, and it occurs in the origin *Org*. The protocol produces output by invoking *receipt* events to the application (higher layer), in the origin *Org* and destination *Dest*.

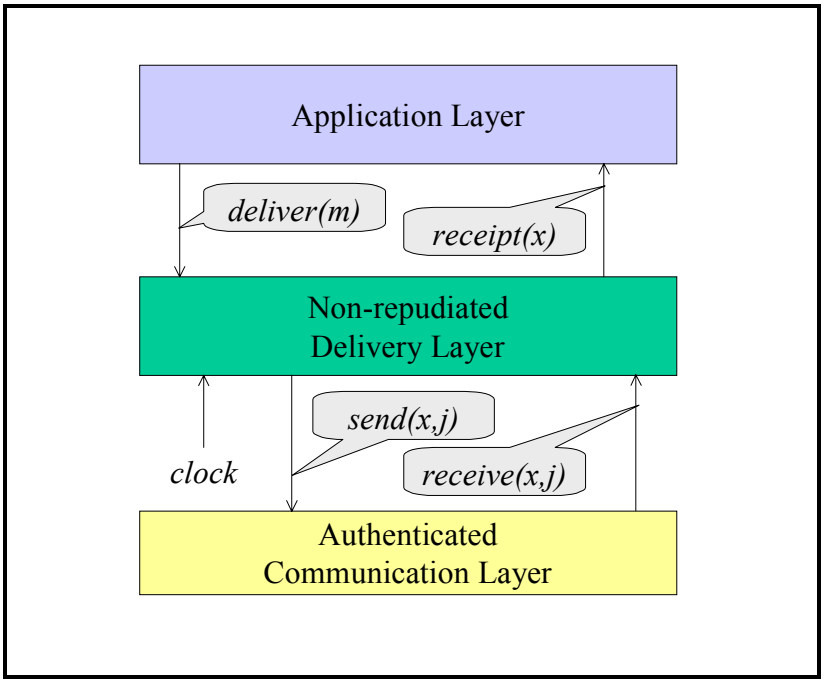

Figure 2: Interfaces of the Non-repudiated Delivery Layer

We assume that processors can communicate using reliable and authenticated communication channels, with bounded delay Δ. Reliability intuitively means that messages sent arrive within Δ, unless the sender or receiver is *faulty*. For simplicity, it is sufficient for our purposes to categorize each processor as either *faulty* or *non-faulty;* non-faulty processors execute the protocol while faulty processors can behave arbitrarily.

To simplify our definitions and analysis, we assume a mapping of all events to real time, and that at any given real time there is at most one input event. We can now define the reliable communication channels properties, for non-faulty processors $i, j \in P$:

- *Secure and reliable send:* if at time t processor i sends $x \in \{0,1\}^*$ to j (i.e. i invokes output event *send(x,j)*), then during time interval $[t,t+\Delta]$, processor j receives x from i (i.e. processor j has input event *receive(x,i)*).
- *Secure and reliable receive:* if at time t processor i receives $x \in \{0,1\}^*$ from j (i.e. i has input event *receive(x,j)*), then during time interval $[t-\Delta,t]$, processor j sent x to i (i.e. processor j invoked output event *send(x,i)*).

To implement reliable and authenticated communication with bounded delay, we may use secure authentication mechanisms in the underlying communication layers, as in Figure 1, based on shared key or public keys[1], e.g. IP-Sec [RFC2411], TLS/SSL [R00,RFC2246] or S/MIME [RFC2633,RFC2634]; or see [CHH00] for proactive secure communication (resilience to and recovery from corruptions). To ensure reliable communication, with bounded delay, use a reliable transport layer such as TCP in conjunction with the authentication mechanism. We notice that such practical mechanism provide only probabilistic security and reliability, and assume computationally bounded adversary; it seems possible to extend our definitions and proofs to allow for this, but for simplicity we adopt the stronger assumptions here.

We also assume that each non-faulty processor has a local clock, synchronized to real time, with maximal drift of Δ. Namely, each non-faulty processor i has a special variable $clock_i$, which is an additional input to the state transition function δ, such that at every time t holds $t-\Delta \leq clock_i \leq t+\Delta$.

To implement local clocks with bounded drift from real time, use reliable and secure clock synchronization protocols, with appropriate (reliable and secure) sources of real-time. The protocol can be an authenticated version of the Network Time Protocol [Mi91,Mi00]. For provable security resilient to corruptions of all clocks, as long as most clocks at any given period are not corrupted, use [BHHN00]. Again, in reality, the bound on the drift is only probabilistic; we believe our results are likely to hold under refined analysis taking this into account, and in this work we use the simpler assumption (that drift is always bounded by Δ).

Designers can use the guaranteed delivery service to ensure non-repudiation of origin and of submission, thereby simplifying the design and analysis of secure e-commerce applications. The guaranteed delivery service cannot completely prevent delivery failures, since there can be a failure in the origin, destination or (one or multiple) delivery authorities. We define the *threshold number of faults* $\boldsymbol{f}$ of a delivery service as the number of delivery authorities that can fail, without disrupting delivery. In executions where the delivery authorities do not fail, or where the number of DA failures is below the permitted threshold f, a non-faulty origin should receive a receipt of submission, and a non-faulty destination should receive the message and receipt of origin. The guaranteed delivery service acts as a `black box`, and the application designer can ignore the delivery authority or authorities, except for validating receipts. For example, we first present protocols for a single Delivery Authority (DA), assuming it does not fail (i.e. $f=0$); then in Section 5 we show how to extend these protocols to utilize multiple DA servers, tolerating up to $f>0$ DA corruptions. However, the designer of an application using the guaranteed delivery service will not need to modify it for using any of the protocols we present, and certainly not to use a different number of authorities and threshold, except possibly for using different validation functions (preferably not even this). A failure of the delivery authorities, beyond the threshold f supported by the guaranteed delivery protocol, is simply mapped to a failure of the origin (if it does not receive receipt) or the destination (if the origin receives receipt but the destination does not receive the message). This is

1 If we use public keys to authenticate the communication, then the Delivery Authority (DA) knows the public keys of the parties, in particular of the origin, and whether the keys are valid (or revoked) at any given time.

justified in the sense that this origin and destination agreed to be held responsible for failure of (more than f) delivery authorities. Our results can be extended to support different fault thresholds for origin and for destination.

We specify a guaranteed-delivery service by defining algorithms for the origin, destination and delivery authority. In addition, the parties agree on a pair of (public) receipt-validation functions $<V_O, V_S>$, for validating receipt of origin and of submission respectively; in some protocols these two functions are identical $V=V_O=V_S$. The agreement on $<V_O, V_S>$ should be public and non-repudiable, e.g. via a contract specifying the functions explicitly, signed by origin and destination. Each receipt-validation function V has two inputs[2]: a message m and a receipt r.

We now define a secure guaranteed delivery service. As an exercise, we first consider a *simplified*, *idealized timing model,* where clocks are completely synchronized, and message transmission is instantaneous[3]. A protocol P, with receipt-validation functions $<V_O, V_S>$ is an *unconditionally secure guaranteed delivery service for up to f faulty delivery authorities under simplified timing model* if:

1. **Valid receipts define origin, destination and time (syntax):** The value of $V(m,r)$, for $V \in \{V_O, V_S\}$, is either $\perp$ (undefined – r is not a valid receipt for m), or a triplet $<Org,Dest,t>$ where *Org* and *Dest* are identities (of origin and destination), and t is time (at which *Org* submitted m to *Dest*).
2. **Receipts are trustworthy (safety): if at most *f* DAs are faulty, then:**
   a. **Receipt of Origin proves delivery from origin, i.e. NRO:** If there is a *receipt(<m,r>)* event in *Dest,* such that $V_O(m,r)=<Org,Dest,t>$, then at time t there was a *deliver(m)* event in *Org ,* or *Org* is faulty.
   b. **Receipt of Submission proves delivery to destination, i.e. NRS:** If there is a *receipt(<m,r>)* event in *Org* such that $V_S(m,r)=<Org,Dest,t>$, then at time t there is a *receipt(<m,r'>)* event in *Dest,* such that $V_O(m,r')=<Org,Dest,t>$, or *Dest* is faulty.
3. **Origin always receives receipt for submission (liveness):** If at most f DAs are faulty, and a *deliver(m)* event occurs in (non-faulty) *Org* at time t, then this is immediately followed by *receipt(<m,r>)* event in *Org*, such that $V_S(m,r)=<Org,Dest,t>$.

We now add details to deal with transmission delays and clock drift. This requires an agreed-upon maximal uncertainty time period M, such that receipts are received within M time units, and[4] the time of submission and reception is bounded within a window of size M. The value of M is derived from the bounds on the delay and on the clock drift; we can compare guaranteed delivery protocols based on how M grows as a function of these bounds, but the application needs only to consider M. A protocol P, with receipt-validation function V, is an *unconditionally secure guaranteed delivery service for up to f faulty delivery authorities with maximal uncertainty M* if:

---

[2] We normally use a keyed function V_k where k is a public validation key, which is also agreed upon between origin and destination as part of the contract they sign. However, since in this paper we only deal with single *<origin, destination>* pair, we consider k as part of the specification of V for simplicity.

[3] For this exercise, ignore our assumption that at any given real time there is at most one input event.

[4] Of course, we could use two different parameters here, but avoid this for simplicity.

1. **Valid receipts define origin, destination and time (syntax):** The value of $V(m,r)$, for $V \in \{V_O, V_S\}$, is either $\perp$ (undefined – r is not a valid receipt for m), or a triplet $<Org,Dest,[min,max]>$ where $[min,max]$ is a time interval (within which *Org* submitted m to *Dest*), where $min<max<min+M$.
2. **Receipts are trustworthy (safety): if at most f DAs are faulty, then:**
    a. **Receipt of Origin proves delivery from origin, i.e. NRO:** If there is a $receipt(<m,r>)$ event in *Dest,* such that $V_O(m,r)=<Org,Dest,[t,t']>$, then during $[t,t']$ there was a $deliver(m)$ event in *Org ,* or *Org* is faulty.
    b. **Receipt of Submission proves delivery to destination, i.e. NRS:** If there is a $receipt(<m,r>)$ event in *Org* such that $V_S(m,r)=<Org,Dest,[t,t']>$, then during $[t,t']$ there is a $receipt(<m,r'>)$ event in *Dest,* such that $V_O(m,r')=<Org,Dest,[t,t']>$, or *Dest* is faulty.
3. **Origin always receives receipt for submission (liveness):** If at most f DAs are faulty, and a $deliver(m)$ event occurs in (non-faulty) *Org* at time t, then during $[t,t+M]$ there is a $receipt(<m,r>)$ event in *Org*, such that $V_S(m,r)=<Org,Dest,[t_0,t_1]>$ and $[t_0,t_1] \subseteq [t-M, t+M]$.

The above definitions are unconditionally secure since they do not allow any probability of forgery, and allow computationally-unbounded attackers. The protocols we present depend on cryptographic mechanisms, which offer only probabilistic guarantees of security, and only against computationally limited adversaries. We therefore now present the final version of our definitions, where we assume computationally limited adversary *ADV* and allow for negligible probability of forgery. Let $ADV(P)$ denote the probability distribution of executions of a protocol P with adversary *ADV,* where adversary controls all the input events and the clock variables, as long as the properties we assumed for the lower layer communication and clock services are kept.

We can now define guaranteed delivery service allowing negligible probability of forgery and assuming computationally bounded adversary. We only need to modify the safety conditions, essentially rephrasing them to state that the forgery events can happen only with negligible probability. A protocol P, with receipt-validation function V, is a *secure guaranteed delivery service for up to f faulty delivery authorities with maximal uncertainty M* if for every poly-time adversary *ADV* holds:

1. **Valid receipts define origin, destination and time (syntax):** The value of $V(m,r)$, for $V \in \{V_O, V_S\}$, is either $\perp$ (undefined – r is not a valid receipt for m), or a triplet $<Org,Dest,[min,max]>$ where $[min,max]$ is a time interval (within which *Org* submitted m to *Dest*), where $min<max<min+M$.
2. **Receipts are trustworthy (safety): if at most f DAs are faulty, then:**
    a. **Receipt of Origin proves delivery from origin, i.e. NRO:** there is negligible probability that an execution in $ADV(P)$ with non-faulty *Org* will contain a $receipt(<m,r>)$ event in *Dest,* such that $V_O(m,r)=<Org,Dest,[t,t']>$, while during $[t,t']$ there was no $deliver(m)$ event in *Org*.
    b. **Receipt of Submission proves delivery to destination, i.e. NRS:** there is negligible probability that an execution in $ADV(P)$ with non-faulty *Dest* will contain a $receipt(<m,r>)$ event in *Org* such that

$V_S(m,r)=<Org,Dest,[t,t']>$, while during $[t,t']$ there was no *receipt*$(<m,r'>)$ event in *Dest,* such that $V_O(m,r')=<Org,Dest,[t,t']>$.

3. **Origin always receives receipt for submission (liveness):** If at most f DAs are faulty, and a *deliver(m)* event occurs in (non-faulty) *Org* at time t, then during $[t,t+M]$ there is a *receipt*$(<m,r>)$ event in *Org*, such that $V_S(m,r)=<Org,Dest,[t_0,t_1]>$ and $[t_0,t_1] \subseteq [t-M, t+M]$.

3. Simple Guaranteed Delivery Protocol

We begin by presenting a simple guaranteed delivery protocol. In this version, we consider a single, completely trusted and reliable Delivery Authority (DA). More specifically, we make the following assumptions:

1. The source and the destination know the public key of the Delivery Authority (DA) PUB_{DA} , and have non-repudiable agreement to honor receipts signed using this key.
2. The Delivery Authority (DA) is completely reliable and trustworthy.
3. The message to be sent is not confidential (in the following section we show how to provide confidentiality).

The flows of the simple guaranteed delivery protocol are illustrated in Figure 3 below. The protocol consists of two request-response pairs, where the Delivery Authority (DA) acts in both as a server. The Delivery Authority (DA) server receives the message from the source in the *Submit request*, and acknowledges it, providing the source with proof of submission, in the *Submit response*. The Delivery Authority (DA) server then keeps the documents, much like existing mailbox servers, until the destination requests them with a *Pickup request.* It is the responsibility of the destination to periodically perform *Pickup requests* (for simplicity, assume that *Pickup requests* should be generated every Δ time units). The Delivery Authority (DA) server responds to the *Pickup request* with the message and proof of origin in *Pickup response*.

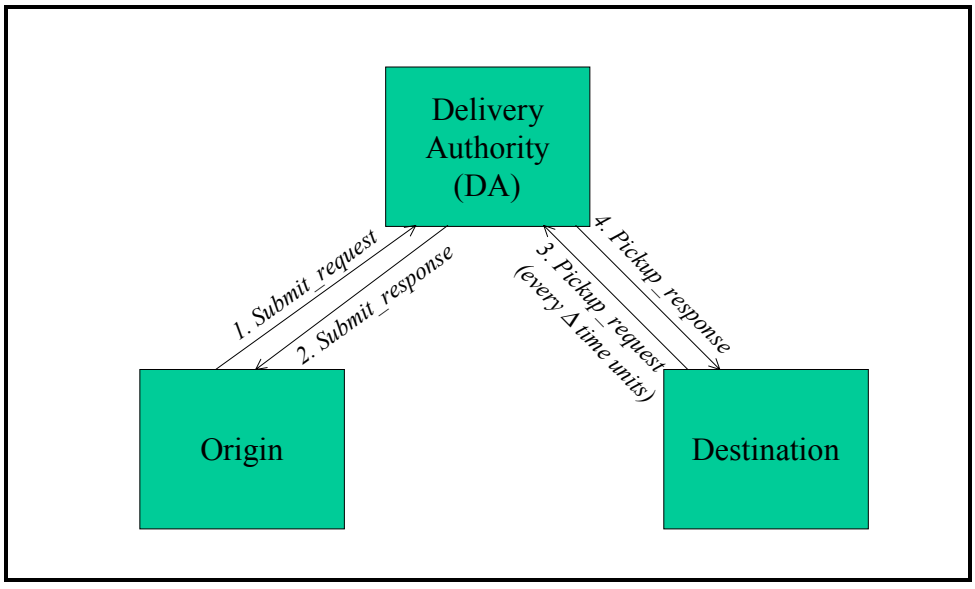

Figure 3: Flows of the Guaranteed Delivery Protocol for a single DA

Let t_{send} denote the (real) time when the origin sends the request. The flows are as follows:

1. *Submit request:* the origin sends $sr=(m,Org,Dest,[clock_{Org}(t_{send})-2\Delta, clock_{Org}(t_{send})+3\Delta])$ to the DA, where m is the message to be delivered, Org is the origin, $Dest$ is the destination, and t_{send} is the (real) time when the origin sends the request (with $clock_{Org}(t_{send})$ being the time available to the origin on its clock).
2. *Submit response:* if the submit request sr is valid, then the DA sends to the origin a receipt (sr,s), where sr is the submit request and s is the DA's signature over it. Otherwise, inform of error.
3. *Pickup request:* the destination sends to the DA a request to pickup message[5] delivered to it via the DA.
4. *Pickup response:* the DA responds to the pickup request by sending the destination the signed receipt (sr,s), where sr is the submit request[6] and s is the DA's signature over it. This is the same signature as used in the submit response.

The security of this protocol relies on the Delivery Authority (DA) correctly verifying the submit request sr. Let t_{rec} denote the (real) time when the request is received at the DA. The Delivery authority should verify[7] that the current time on its clock, $clock_{DA}(t_{rec})$, is within timestamp interval in sr, say $[t,t']$. For non-faulty origin, $t=clock_{Org}(t_{send})-2\Delta$ and $t'=clock_{Org}(t_{send})+3\Delta$. The delay is at most Δ, i.e. $t_{send} \leq t_{rec} \leq t_{send}+\Delta$. The maximal clock drift is also Δ; therefore:

- $clock_{DA}(t_{rec}) \leq c+\Delta \leq t_{send}+2\Delta \leq clock_{Org}(t_{send})+3\Delta = t'$
- $clock_{DA}(t_{rec}) \geq t_{rec}-\Delta \geq t_{send}-\Delta \geq clock_{Org}(t_{send})-2\Delta = t$

This shows the following claim, which is the key to the proof of liveness:

Claim 1: a non-faulty DA will never reject submit requests sent correctly by a non-faulty origin *Org,* and will always send a receipt to the source (immediately in *Submit response*) and forward them to the destination (on next *Pickup response*) .

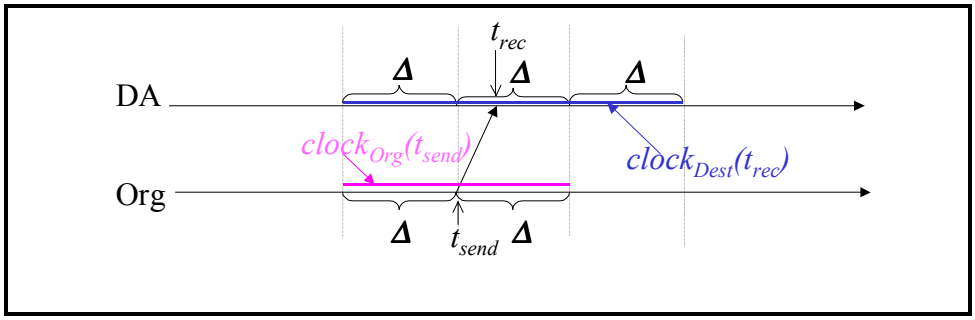

Figure 4: Timing of requests from non-faulty origin

The validation function V is the same for receipt of origin (NRO) and of submission (NRS), $V=V_O=V_S$, with input (m,r). It validates that r is a valid receipt for m, i.e. that $r=(sr,s)$, with $sr=(m,Org,Dest,[t,t'])$ and that s is a signature by the DA on sr. The

[5] For simplicity, we assume there is only one message for any pickup request. If multiple messages (and origins) may be delivered to the destination by the same DA, the pickup request can specify the origin(s), and/or the pickup reply may contain multiple deliveries. The details are simple and omitted.
[6] The pickup response may contain multiple messages (submit requests).
[7] The DA should also verify that the origin has sent the submit request, but we assumed this is done by the communication layer and therefore do not discuss it explicitly. This check may use shared key or public key mechanisms.

public key of the DA *(PUB_{DA})*, used for validating that *s* is a signature by DA on *sr*, is part of the definition of *V*. If *r* is a valid receipt for *m,* then *V* outputs triplet *<Org,Dest,[t,t']>*; otherwise it outputs ⊥. Clearly:

Claim 2: Whenever a non-faulty DA sends receipt *(sr,s)*, following steps 2 and 4 of the protocol above, then:

1. *Syntax: sr=(m,Org,Dest,[t,t'])* for some message *m* and time interval *[t,t']*.
2. *Validity: V(m,(sr,s))= <Org,Dest,[t,t']>*

We can now prove:

Theorem 1. The Simple Guaranteed Delivery protocol with receipt-validation function *V,* is a *secure guaranteed delivery service for non-faulty faulty DA, with maximal uncertainty 5Δ.*

Proof: The syntax property follows from Claim 2. Liveness follows from Claim 1, and the validity property of Claim 2 (notice that the delay is bounded by Δ and *M=5Δ*). It remains to prove safety.

Assume, to the contrary, that *ADV* is a poly-time adversary with significant probability of generating execution with *receipt(<m,r>)* event in *Dest,* such that *V(m,r)=<Org,Dest,[t,t']>,* while during *[t,t']* there was no *deliver(m)* event in non-faulty *Org.* Therefore, in particular, there is no *deliver(m)* even at time *t''* such that *t''=t+2Δ* and *t''=t'-3Δ;* therefore non-faulty *Org* does *not* send *sr=(m,Org,Dest,[t,t'])* to DA. From the *secure and reliable receive* property of the communication channel, it follows that the DA does not receive *sr* from *Org.* It follows that the DA never signs *sr.* But from the definition of *V,* clearly *r=(sr,s)* where *s* is a valid signature using the DA's private key on *sr.* It is therefore possible, with significant probability, to use *ADV* to generate a signature using the DA's private key on a message that the DA never signed, without access to the DA's private key, contradicting the definition of a secure signature scheme. This proves safety of NRO.

Safety of NRS follows similarly, from the *secure and reliable send* property of the communication channel. ■

Notice that the simple guaranteed delivery protocol does not use any public key operation by the origin; in particular the origin does not sign the message or the delivery request. This may seem surprising, as non-repudiation of origin is usually associated with public key signature by the origin. Adding signature by the origin may, indeed, ensure safety for NRO even if the DA fails (arbitrarily); we discuss this in Section 5, along with more advanced mechanisms for guaranteed delivery in the presence of faults (and recoveries).

4. Guaranteed Confidential-Message Delivery

In the previous section, we assumed that the message to be sent is not confidential. In fact, using the protocol of the previous section, it suffices that the parties trust the Delivery Authority regarding the confidentiality of the message as well as regarding the receipts (in which case, we could use the secure communication mechanisms we

assumed that the DA has with both origin and destination, to hide the requests and responses in transit). In this section, we outline how to extend the protocol in order to protect the confidentiality of the message, without trusting the DA in this matter.

The receipt which the origin receives from the DA is timestamped, and therefore the origin can use the DA to time-stamp a document by sending it to an arbitrary destination (e.g. itself); this is similar to using certified postal services for time-stamping documents in a physically sealed envelop. Several proposals were made for preserving confidentiality of a timestamped document even from the trusted authority providing the timestamp (notary, time stamping authority, or in our case delivery authority). In existing proposals for time stamping, e.g. [HS91], this is achieved by time-stamping the hash of the message, using one-way, collision resistant hash function. This is a natural heuristics, however it is not secure under standard definitions of one-way collision resistant hash functions; for example it may be easy to find out parts of the message. One may use oracle hash functions [C97,CMR98], but notice even these allow an adversary to compare the hash to hash of a particular message; this may still be an exposure for some applications.

To describe our solution, we first observe that the delivery authority (DA), unlike a time-stamping service, must also forward the message to the destination. The natural way to achieve this is that the origin will encrypt the message using the destination's public key, i.e. send to the delivery authority $Enc_{Dest}(m,z)$, where z are any random bits required for encryption.

Encrypting a message and then signing it (*EtS)* conflicts with the widely accepted design principle `*Sign then Encrypt*` (*StE*) of [AN95, AN96,Sy96]. We agree that one must be cautious in signing an encrypted document. This is certainly true of applications (like ours) where the signer receives the document already encrypted. In fact, even if the signer is also doing the encryption, *EtS* may not provide non-repudiation. In fact, it is easy to show an (artificial) example of secure encryption and signature schemes, such that if the origin encrypts a message for the destination and then signs the resulting ciphertext, then the destination may be able to produce a different message and public key, that encrypt to the same ciphertext, and therefore to forge a signature. So how can we sign the encrypted document?

To solve this, the Delivery Authority (DA), signs, in addition to the ciphertext, also the public key of the destination, PUB_D. For the case where signing and encryption are by the same party, [ADR02] show that this method of using *EtS* is secure, when using cryptosystems where the random bits z applied during encryption are recovered during decryption (as with most cryptosystems). While in our case the origin encrypts and the DA signs, we believe that the proof of [ADR02] can be extended to cover this as well.

It is also possible to use a public key cryptosystem where the random bits used for encryption are not recovered during decryption. In this case, the DA signs, in addition, a *commitment c* to the message *m,* computed by *Org*. Namely, c is the computed from $<c,d>=Commit(m)$, where $<Commit, Decommit>$ is a secure *commitment scheme*[8]. Efficient and provable[9] secure constructions for commit

[8] More precisely, we use a commitment scheme with a public key, agreed upon between origin and destination as part of their contract; but since this public key is fixed for our discussion (which

schemes appear in [DPP94, DPP98, HM96]. The properties of the commitment scheme are:

1. If $<c,d>=Commit(m)$, then $m=Decommit(c,d)$.
2. Given $<c,d>=Commit(m)$, it is infeasible to find a *collision,* i.e. $m' \neq m$ and d' such that $m'=Decommit(c,d')$.
3. The commitment c gives no information about the input m.

By signing the commitment rather than the actual message, the message is hidden from the DA. The destination can prove that it received message m by presenting the signed receipt together with the decommitment d. Since d is necessary and sufficient to recover m, we send encryption of d rather than of the message m. We call this method for providing non-repudiation via a third party on hidden messages *Commit then Encrypt then Sign (CtEtS)*[10]. The signature together with the decommitment provides the destination with a proof of origin

For proof of submission, the origin must also prove that it provided the ciphertext correctly, so that the destination is able to decrypt and retrieve the message m. If the encryption function is deterministic, it is sufficient for the origin to provide d, since it is possible then to compute the ciphertext by applying the encryption function to d and comparing the signed ciphertext value. It is also easy to extend this to allow for probabilistic encryption (as required for security); the origin should simply produce the random bits z used during the encryption process, i.e. the encrypted message is $x=E_D(d,z)$.

The protocol uses the same flows as in Figure 3. We only modify the *Submit request* as follows. The origin sends $sr=(<x,PUB_D,c>,Org,Dest,[t-2\Delta,t+3\Delta])$ to the DA, where *m, Org, Dest,* and t are as before, and:

1. $<c,d>=Commit(m)$.
2. PUB_D is the public key of the destination
3. $x=E_D(d,z)$ is encryption of d, using public key of the destination PUB_D and random bits z.

The Delivery Authority (DA) should correctly verifying the submit request, performing the same checks as for the simple guaranteed delivery protocol (in previous section), and in addition validating that PUB_D is a valid public key of the destination (in particular, that it was not revoked).

We need different receipt validation functions V_S and V_O for proving submission and origin, respectively; as before, the public key of the DA (PUB_{DA}) is part of the definition of both validation functions. The origin validation function V_O has input (m,r), where m is the message and $r=(sr,s,d)$. Its output is the triplet

---

considers only a single pair of <origin, destination>, we consider the public key as part of the definition of the commitment scheme, for simplicity (similarly to our definition of V).

[9] Many practical protocols and systems use $Commit(m)=h(m,d)$, where d chosen randomly. This heuristic is secure under the random oracle model, but not under specific choices of h; the provable constructions are nearly as efficient and hence preferable.

[10] Notice *CtEtS* is different from the *Commit then Encrypt* ***and*** *Sign (CtE&S)* method proposed to allow parallel computation of encryption and signature in [ADR02].

<*Org,Dest,[t',t'']*> taken from *sr*. This output is produced only if the following validations are all successful:

1. *s* is a signature by the DA (i.e. verified using PUB_{DA}) on *sr,*
2. *m=Decommit(c,d)* where *c* is taken from *sr,*
3. The public key of the DA (PUB_D) is correctly included in *sr*.

The submission validation function V_S has *z* as an additional input, i.e. *r =(sr,s,d,z).* It is exactly like V_O but with an additional validation:

4. $x=E_D(d,z)$.

5. Delivery Authority Corruptions and Recoveries

Guaranteed delivery requires the origin and destination to place considerable trust in the Delivery Authority (DA). The DA is responsible for authenticating the messages from the origin, validating the timestamp in the messages, delivering the messages to the destination, and – for confidential messages – validation of the destination's public key. A failure of any of these functions, malicious or benign, can appear to the application as a failure of either origin or destination, possibly resulting in penalties as per the agreement between them.

In the following subsections we sketch two extensions of the protocols that provide (different levels of) resiliency to corruptions of DA servers. In the first subsection we discuss the extra security gained when the origin signs the delivery request sent to the DA; this is limited to safety of non-repudiation of origin. In the second subsection, we discuss the use of multiple, redundant delivery authorities, which allows resiliency to corruptions of authorities, as long as at least some threshold number of authorities is not corrupted (in total, or during any given period if we allow recovery from corruptions).

5.1. Safe Non-Repudiation of Origin with Corrupted DA

The dependency on the trustworthiness of the DA is especially evident regarding the non-repudiation of origin. Non-repudiation of origin is often considered an obvious application for digital signatures. We also use digital signatures for non-repudiation of origin; however the traditional approach is for the origin to sign the message, while in the protocols presented so far (only) the DA signs the message. We now discuss briefly the value of adding the origin's signature on the request *sr*, in addition to the DA's signature.

For the origin's signature to be meaningful, the agreement between the origin and destination (as reflected in the validation functions) must identify the origin's public key. The public key may be identified directly in the agreement. Alternatively, the public key may be signed by a trusted certificate authority (CA), and the agreement identifies the CA (and in particular includes the CA's public key).

In any case, the public key should remain valid throughout any possible dispute between the origin and destination. Otherwise, the origin can claim that the signature was performed using a revoked or expired key. We could time-stamp the request, but

this will again introduce dependency on the DA (or another third party). To protect against a corrupted DA backdating the signature on the timestamp, use a forward-secure signature algorithm as in [BM99].

With this extension, we ensure the safety of the proof of origin even if the DA can be corrupted. Namely, If *Dest* has <*m,r*> such that $V_O(m,r)$=<*Org,Dest,[t,t']*>, then *Org* submitted message *m* to *Dest* at time between *t* and *t',* or *Org* is faulty.

5.2. Multiple Delivery Authorities Protocol, Resilient to Corruptions and Recoveries

It is therefore natural to consider how to provide the origin and destination with a highly trustworthy guaranteed delivery service, using realistic assumptions on the delivery authority servers. In particular, it is desirable to design the guaranteed delivery service using multiple delivery authorities, in such a way that as long as the number of corrupted authorities is below some predefined threshold, the service is trustworthy. It is even better if the service is trustworthy even when each server may become corrupted, as long as the number of servers corrupted during any time period (of given length) is below some predefined threshold.

We can achieve resiliency to corruption of delivery authorities by using multiple, redundant DA design as in Figure 5. The flows of this multiple-DA protocol are exactly as described in the previous sections, except that multiple requests and responses are sent for each delivery (corresponding to the multiple delivery authorities).

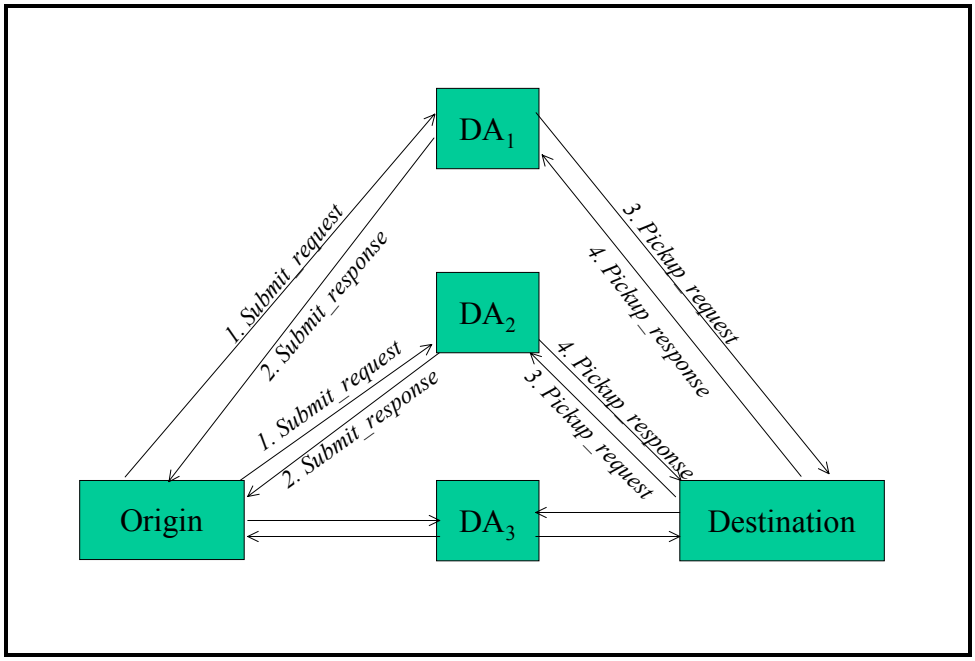

Figure 5: Corruption-resiliency by multiple, redundant Delivery Authorities (DA)

We define the validation function for a particular DA, say DA_i, denoted $V_{O,i}, V_{S,i}$ exactly as V_O, V_S as in the previous sections, using the public keys of DA_i. The validation function V_O for the multiple-DA protocol are defined by the threshold

parameter f as follows:

$$V_O(m,r) = \begin{cases} < Org, Dest, [t,t'] > & if\ r = \{(i,r_i)\} s.t. |\ r\ | > f\ and \\ & (\forall i) V_{O,i}(m,r_i) =< Org, Dest, [t,t'] > \\ \\ \perp & Otherwise \end{cases}$$

Validation function V_S is defined similarly.

We continue the discussion focusing on the case where each DA carries the simple guaranteed delivery protocol of Section 3. Claims 1 and 2 still hold, when restated for one DA out of the set:

Claim 1': a non-faulty DA, say DA_i, will never reject submit requests sent correctly by a non-faulty origin *Org,* and will always send a receipt to the source (immediately in *Submit response*) and forward them to the destination (on next *Pickup response*) .

Claim 2': Whenever a non-faulty DA, say DA_i, sends receipt *(sr,s)*, following steps 2 and 4 of the protocol, then:

1. *Syntax: sr=(m,Org,Dest,[t,t'])* for some message *m* and time interval *[t,t']*.
2. *Validity: V(m,(sr,s))= <Org,Dest,[t,t']>*

And from this follows:

Theorem 2. The Multiple-DA Guaranteed Delivery protocol with receipt-validation function *V,* is a *secure guaranteed delivery service for up to f faulty delivery authorities, with maximal uncertainty 5Δ*.

Proof: Similar to proof of Theorem 1. ■

The Multiple-DA protocol as presented above has the undesirable property that the validation process requires the public keys of all the delivery authorities which generated the receipts, and in particular awareness of the entity performing the validation to the value of the threshold *f* and to the identities of the delivery authorities. Validation also requires at least *f* signature verification operations. We can avoid these drawbacks, by using a threshold signature scheme, e.g. [S00], since *all the delivery authorities sign exactly the same message.* This provides a `transparent` solution, where validation of receipts of origin and of submission are exactly the same for a Multiple-DA protocols as it is for a single DA protocol.

Using the fact that all the delivery authorities sign exactly the same message, we can also use proactive signature schemes, e.g. [HJJ*97,R98], to allow resiliency to failures of every DA, as long as the number of corrupted delivery authorities during any time period of specific length Π (with $\Pi > M$) is bounded by *f.* This requires, of course, that the underlying communication and clock synchronization mechanisms are also proactively secure, e.g. using the protocols of [BHHN00,CHH00].

6. Applications of Guaranteed Delivery

We now discuss how some important secure electronic commerce applications can take advantage of an underlying guaranteed delivery service, as illustrated in **Figure 1**.

Consider first a simple electronic banking and brokerage application as illustrated in Figure 6. The client of the banking or brokerage service sends an order or request to the server, e.g. to buy a particular security. If the bank (or broker) is not available to perform the service, the customer may suffer losses; therefore it is reasonable for the customer and the bank to agree in advance that the bank is committed to receive orders sent by the customer thru a guaranteed delivery channel, e.g. certified post. The guaranteed delivery service facilitates such agreements via a communication network.

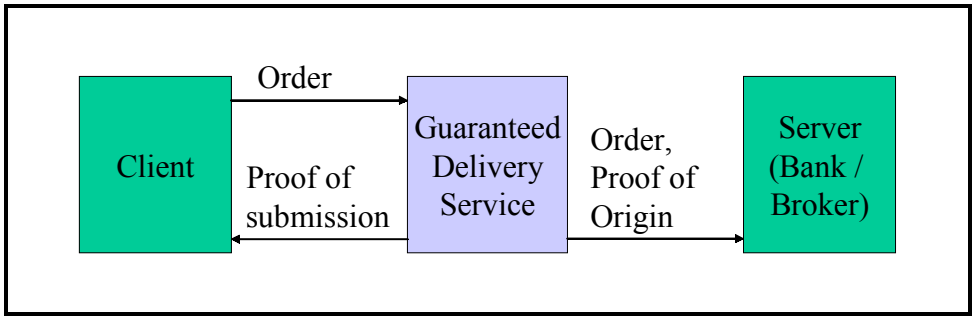

Figure 6: Electronic Banking / Brokerage Application of Guaranteed Delivery

A similar scenario exists in many business-to-business applications, such as supply chain relationships between a buyer and a provider, as in Figure 7. Here, both parties may need receipts for submission of requests as well as of replies.

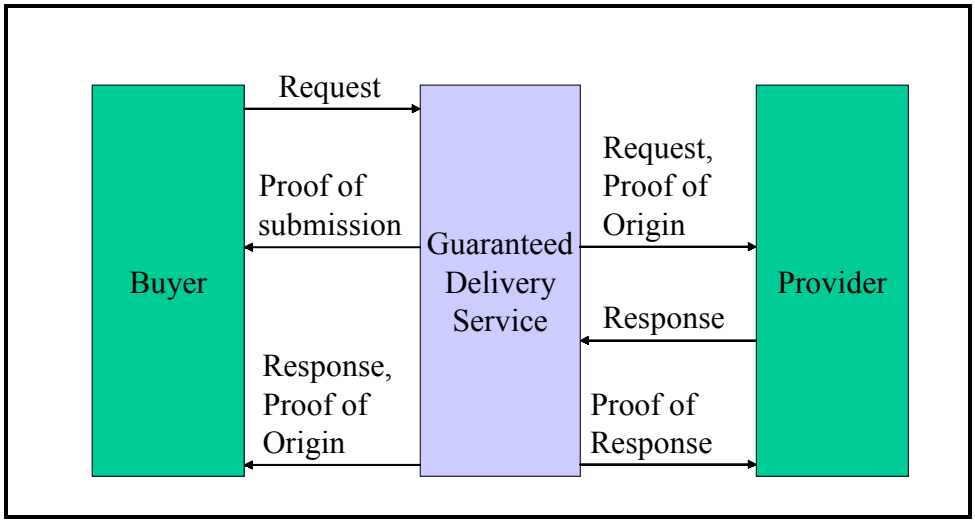

Figure 7: Business to Business Supply Chain Application of Guaranteed Delivery

We next illustrate applicability of guaranteed delivery to payment protocols. Specifically, the `Final payments` protocol of [H02b], illustrated in Figure 8. In this protocol, each Payment Service Provider (PSP), e.g. PSP_B, periodically sends signed *Payment Routing Tables (PRT)* in which it commits to honor *Payment Orders (PO)* signed by other PSPs (e.g. PSP_A). However, the PRT specifies `*extra time*`, such that a PO must be deposited this extra time before it expires (to allow PSP_B to deposit the PO at PSP_A in time). This assumes a guaranteed delivery mechanism provides a

timestamped proof of submission (for deposit of POs), so that payments are assured even if the PSP is (temporarily) disconnected. This application does not use the proof of origin, as the PO is signed by the issuing PSP.

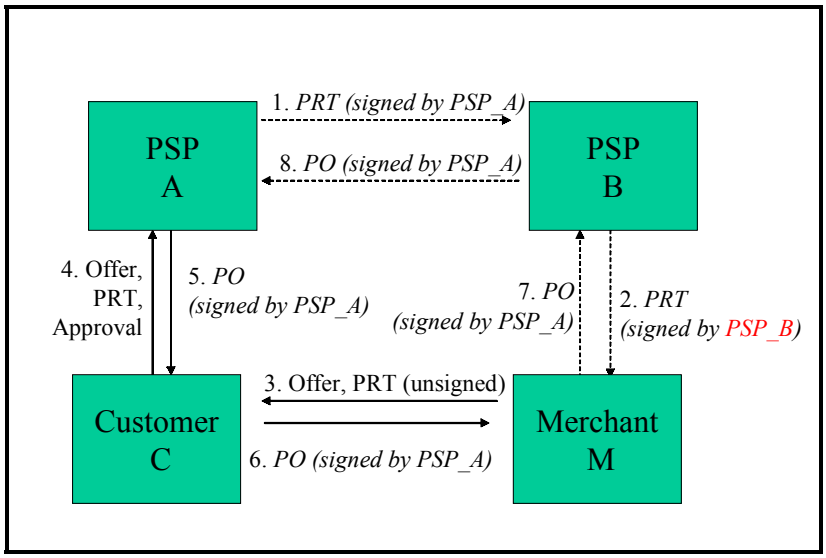

Figure 8: Final Payments Protocol using Guaranteed Delivery

Finally, we mention briefly that guaranteed delivery is useful also as underlying layer for secure bidding, gambling and `closed` auctions applications. In such applications, the client often submits some bid (bet or offer, respectively). The client wants a receipt to prove its submission; the content of the bid (bet or offer) often have to be kept confidential from the server (and other clients) until a predefined deadline. The delivery authority can support such applications, with the trivial extension of forwarding all bids (bets or offers, respectively) to the destination only at the predefined deadline, with a proof of origin that may also show this is the complete set of bids (bets or offers, respectively). See illustration in Figure 9.

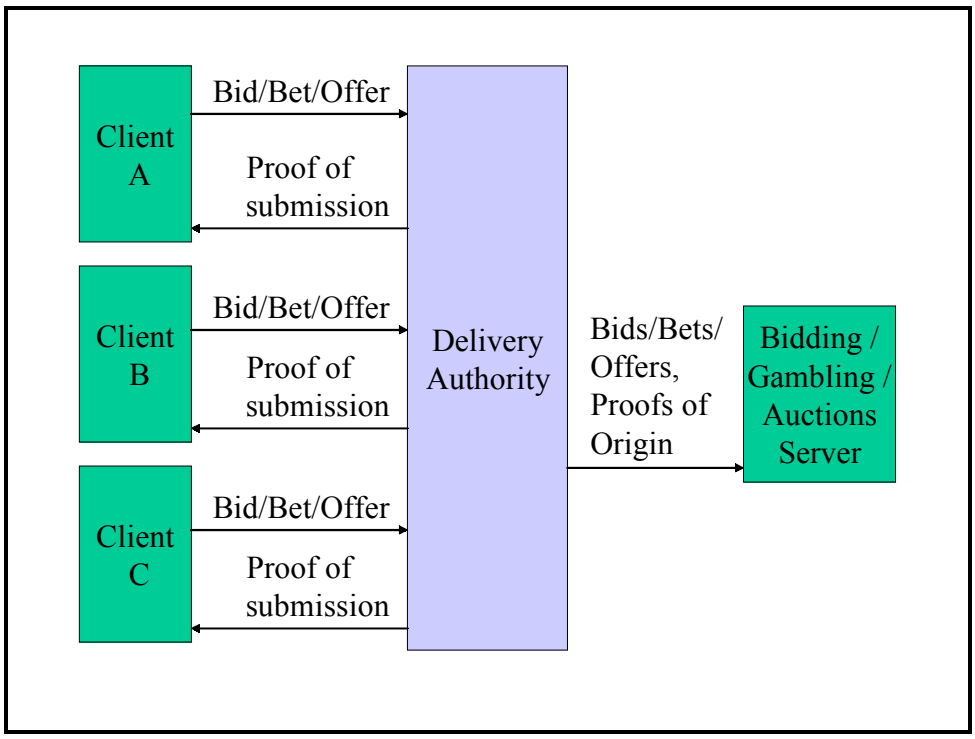

Figure 9: Using Guaranteed Delivery for Bidding, Gambling and Auctions

7. Optimistic Protocols

A well-known design principle for fault-tolerant and secure systems is: for safety, assume the worst; for optimization, be optimistic. Systems and protocols optimized for the typical case, with no faults and delays much lower than the timeout bounds [HK00], are often called *optimistic*. Several works study optimistic certified delivery protocols, e.g. see [ASW97,KMZ02, M97, PSW00, Z01, ZG96]; these works avoid the involvement of the DA in typical, fault-free executions[11]. Several of these protocols have the appealing property that the proofs (of origin and of receipt) of executions where the DA is involved are indistinguishable from the proofs in executions where the DA is offline; we say that these protocols are *transparent*.

It is easier to design optimistic guaranteed delivery protocols. Specifically, assume that the origin and destination agree on (fixed) public keys, without allowing revocations. Then the origin can try to submit a signed and time-stamped message directly to the destination, which should send back a signed receipt. Only when there is a failure, e.g. the origin does not receive the signed receipt (e.g. by 2Δ), would the origin send the message through the DA (or multiple DA's).

8. Conclusions and Further Research

Non-repudiation is a basic requirement from commercial relationships, especially in for business-to-business commerce and for banking and payment applications. We show that many of these applications can be built on top of a relatively simple and efficient service of *guaranteed delivery*. Other e-commerce applications may require other non-repudiation services, such as certified delivery, which is studied by other

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

works; we propose that as a general architecture, one may consider a *non-repudiation delivery layer* as an underlying service to many (most?) secure e-commerce applications.

Our work is trying to provide solid foundations for practical electronic commerce protocols and systems. We expect further work to provide additional solid mechanisms for secure e-commerce applications, especially in the banking and finance areas, where the importance of security is well recognized.

There are many directions for further research based directly on the current work, such as implementation, standardization, deployment and applications. In addition, let us mention one specific area that was not addressed, namely, support for interoperability between origin and destination that use (and trust) different delivery authorities. This is important, in particular to support ad-hoc relationships between potential business partners, which may not justify the effort to agree on a common set of delivery authorities. A solution may employ trust relationships between multiple delivery authorities, using hierarchy or a `web of trust`.

References

[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.

[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. http://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. http://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.

[BHHN00] Boaz Barak, Shai Ha-Levi, Amir Herzberg, Dalit Naor, Clock Synchronization with Faults and Recoveries, proceedings of the Nineteenth ACM Symposium on Principles of Distributed Computing (PODC 2000), July 16-19 2000, Portland, Oregon, pp. 133-142.

[BM99] Mihir Bellare and Sara K. Miner. A Forward-Secure Digital Signature Scheme. In Proc. of Crypto, pp. 431--448, 1999.

[C97] Ran Canetti, “Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information”, Advances in Cryptology – Crypto 97 Proceedings, pp. 455-469, 1997.

[CHH00] Ran Canetti, Shai Ha-Levi and Amir Herzberg. "Maintaining authenticated communication in the presence of break-ins". In Journal of Cryptography, Vol. 13, No. 1, January 2000, pp. 61-105. Extends version in Proceedings of the sixteenth annual ACM symposium on Principles Of Distributed Computing (PODC), 1997, Pages 15 - 24.

[CMR98] R. Canetti, D. Micciancio and O. Reingold, "Perfectly One-Way Probabilistic Hash Functions," Proceedings of 30th STOC, 1998.

[DPP94] Ivan B. Damgård, Torben P. Pedersen, Birgit Pfitzmann: On the Existence of Statistically Hiding Bit Commitment Schemes and Fail-Stop Signatures; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 250-265.

[DPP98] Ivan B. Damgård, Torben P. Pedersen, Birgit Pfitzmann: Statistical Secrecy and Multi-Bit

Commitments; IEEE Transactions on Information Theory 44/3 (1998) 1143-1151.

[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.

[H02a] Amir Herzberg, Securing XML, Dr. Dobbs Journal, March 2002.

[H02b] Amir Herzberg, Introduction to Secure Communication and Commerce using Cryptography, Chapter 12 – Payments, draft of book to be published by Prentice-Hall, available online at http://amir.herzberg.name/book.html.

[HJJ*97] Proactive public key and signature systems; Amir Herzberg, Markus Jakobsson, Stanislav Jarecki, Hugo Krawczyk, and Moti Yung; Proceedings of the 4th ACM conference on Computer and communications security , 1997, Pages 100 – 110.

[HK00] A. Herzberg and S. Kutten. “Early Detection of Message Forwarding Faults”, SIAM Journal of Computing, August-October issue, 2000.

[HM96] S. Halevi and S. Micali, "Practical and Provably-Secure Commitment Schemes from CollisionFree Hashing", in Advances in Cryptology - CRYPTO96, Lecture Notes in Computer Science 1109, Springer-Verlag, 1996, pp. 201-215. http://citeseer.nj.nec.com/halevi96practical.html

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

[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 http://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", partially available at citeseer.nj.nec.com/lacoste00semper.html.

[Mi00] Mills, D.L. Public key cryptography for the Network Time Protocol. Electrical Engineering Report 00-5-1, University of Delaware, May 2000. 23 pp. Available online from http://www.eecis.udel.edu/~mills/bib.htm.

[Mi91] Mills, D.L. Internet time synchronization: the Network Time Protocol. IEEE Trans. Communications COM-39, 10 (October 1991), 1482-1493. Available online from http://www.eecis.udel.edu/~mills/bib.htm.

[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.

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

[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.

[RFC1939] J. Myers, M. Ross, Post Office Protocol - Version 3, IETF Network working group, August 2001, http://www.ietf.org/rfc/rfc3161.txt.

[RFC2246] T. Dierks, C. Allen, The TLS Protocol: Version 1.0, Network Working Group, Internet Engineering Task Force (IETF). Available online at http://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 http://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, http://www.ietf.org/rfc/rfc2560.txt.

[RFC2633] B. Ramsdell, Editor, S/MIME Version 3 Message Specification, Network Working Group, Internet Engineering Task Force (IETF). Available online at http://www.ietf.org/rfc/rfc2633.txt. June 1999.

[RFC2634] P. Hoffman, Editor, Enhanced Security Services for S/MIME, Network Working Group, Internet Engineering Task Force (IETF). Available online at http://www.ietf.org/rfc/rfc2634.txt. June 1999.

[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, http://www.ietf.org/rfc/rfc3161.txt.

[S00] Victor Shoup, "Practical Threshold Signatures", Advances in Cryptology - Eurocrypt 2000, Springer-Verlag LNCS 1807, pp.207-220, 2000.

[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. http://citeseer.nj.nec.com/syverson96limitation.html

[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. http://citeseer.nj.nec.com/28044.html

Appendix A: Symbol Tables

DA	Delivery Authority
Org	Origin
Dest	Destination
m	Message to deliver
$\perp$	Undefined ($V(m,r)=\perp$ if r is not valid receipt of m)
r	Receipt
V(m,r)	Receipt validation function, $<o,d,[t,t']>$ or $\perp$
V_S	Receipt validation function for proof of submission
V_O	Receipt validation function for proof of origin
M	Maximal uncertainty time period, agreed between parties
Δ	Bound on message delay and clock drift
PUB_D	Public key of Destination (*Dest*)
PUB_{DA}	Public key of Delivery Authority (DA)
sr	Submit request, *sr=(m,o,d,[t,t'])*
c	Commit value
d	Decommit value
z	Random bits used for encryption

Table 1: List of symbols (part I)

P	The set of processors; includes *Org, Dest*
Deliver	Event where application provides message *m* (once, in the origin)
Receipt	Event where a receipt (and message) is provided to the application
f	Maximal number of faulty delivery authorities
$Clock_p(t)$	Value of the clock of processor *p* at real time *t*
$\mathcal{P}$	Protocol, i.e. mapping from *P* to tuples *(IE, OE, S, init, δ)*.
IE	Input events
OE	Output events
S	States (of a processor)
init	Initial state ($init \in S$)
δ	State transition function
$V_{O,i}, V_{S,i}$	Validation functions for proof of origin (resp. submission) for DA_i
DA_i	The i^{th} delivery authority (for multiple-DA protocol)
t_{rec}	(real) time that request is received at DA
t_{send}	(real) time that request is send to DA

Table 2: List of symbols (part II)