*International Journal of Research and Computational Technology Vol.2 Issue.3*

ISSN: 0975-5665, June, 2012 www.ijrct.org

# Research on Secure Distributed Online Certification Authority

**# V.Mamatha[1],E mail**: *mamathareddy.vudem@gmail.com*
**# P.Kumara Swami [2]** *palleboina.kumar@gmail.com*
**# Dr.C.V.Gururao[3]** professor & Head of the department
Department of Computer Science and Engineering
**# S.R Engineering College, Warangal, A.P, INDIA**

## Abstract:

***A fault-tolerant and secure online certification authority that has been built and deployed both in a local area network and in the Internet. Extremely weak assumptions characterize environments in which COCA's protocols execute correctly: no assumption is made about execution speed and message delivery delays; channels are expected to exhibit only intermittent reliability; and with 3t C 1 COCA servers up to t may be faulty or compromised. COCA is the first system to integrate a Byzantine quorum system (used to achieve availability) with proactive recovery (used to defend against mobile adversaries which attack, compromise, and control one replica for a limited period of time before moving on to another). In addition to tackling problems associated with combining fault-tolerance and security, new proactive recovery protocols had to be developed. Experimental results give a quantitative evaluation for the cost and effectiveness of the protocols.***

*Categories and Subject Descriptors: C.2.0 [Computer-Communication Networks]: General—Security and protection; C.2.4 [Computer-Communication Networks]: Distributed Systems—Client/server; D.4.5 [Operating Systems]: Reliability—Fault-tolerance; D.4.6 [Operating Systems ]: Security and Protection—Authentication; cryptographic controls; E.3 [Data]: Data Encryption—Public key cryptosystems*

## 1. INTRODUCTION

In a public key infrastructure, a *certificate* [Kornfelder 1978] specifies a binding between a name and a public key or other attributes. Over time, public keys and attributes can change: a private key might be compromised, leading to se-lection of a new public key, for example. The old binding and any certificate that specifies that binding then become *invalid*. A *certification authority* (CA) attests to the validity of bindings by issuing digitally signed certificates that specify these bindings and by providing a means for clients to check the validity of certificates. With an online CA, principals can check the validity of certificates just before using them. COCA (Cornell On-Line Certification Authority), the subject of this article, is such an online CA.

COCA employs replication to achieve availability and employs proactive recovery with threshold cryptography for digitally signing certificates in a way that defends against *mobile adversaries* [Ostrovsky and Yung 1991] which at-tack, compromise, and control one replica for a limited period of time before moving on to another. In that, the system is not novel. What distinguishes COCA is its qualitatively weaker assumptions about communication links and execution timing. Many denial-of-service attacks succeed by invalidating stronger communication and execution-timing assumptions; in making weaker assumptions, COCA is less vulnerable to these attacks.

New proactive recovery protocols had to be developed for execution in this relatively unconstrained and more realistic environment. Moreover, because implementing agreement is problematic in the absence of execution-timing assumptions [Fischer et al. 1985], COCA employs a Byzantine quorum sys-tem [Malkhi and Reiter 1998a] (rather than the state machine approach [Lamport 1978]) for managing replicated state. In so doing, COCA is the first to tackle the problems associated with integrating threshold cryptography and Byzantine quorum systems. Thus, beyond its intrinsic utility for public key infrastructures, COCA has pedagogical value as a vehicle for understand-ing how to combine mechanisms for supporting fault-tolerance and security properties.

Besides its weak assumptions, a variety of traditional means for combating denial-of-service attacks is used by COCA: (i) processing only those requests that satisfy authorization checks, (ii) grouping requests into classes and mul-tiplexing resources so that demands from one class cannot have an impact on processing of requests from another, have an impact on as well as (iii) caching results of expensive cryptographic operations. And although resource-clogging denial-of-service attacks certainly remain possible, experiments demonstrate that launching a successful attack against COCA is harder with these mechanisms in place. In fact, simulated denial-of-service attacks have allowed us to measure the effectiveness of the various means COCA employs to re-sist denial-of-service attacks, so the work reported herein contributes much-needed experimental data on the performance of traditional denial-of-service defenses.

## 2. SYSTEM MODEL AND SERVICES SUPPORTED

COCA is implemented by a set of $n$ servers, each running on a separate processor in a network. We intend COCA for use in an environment like the Internet. Thus COCA tolerates failures and defends against malicious attacks, subject to the following assumptions.

*Servers*. Servers are either *correct* or *compromised*, where a compromised server might stop executing, deviate arbitrarily from its specified protocols (i.e., Byzantine failure), and/or disclose information stored locally. System execution is viewed in terms of protocol-defined periods called *windows of vulnerability*; terms "correct" and "compromised" are relative to those periods. Specifically, a server is deemed correct in a window of vulnerability if and only if that server is not compromised throughout that period. We assume:

—at most $t$ of the $n$ COCA servers are ever compromised during each window of vulnerability, where $3t$ C $1 \cdot n$ holds;

—clients and servers can digitally sign messages using a scheme that is existentially unforgeable under adaptively chosen message attacks;

—various cryptographic algorithms (e.g., public key cryptography and thresh-old cryptography) COCA employs are secure.

*Fair Links*. A *fair link* is a communication channel that does not necessarily deliver all messages sent, but if a process sends infinitely many messages to a single destination then infinitely many of those messages are correctly delivered. In addition, messages in transit may be disclosed to or altered by adversaries.

The communications network is assumed to provide (only) fair links. (With-out some comparable assumption about the network, an adversary could pre-vent servers from communicating with each other or with clients.)

*Asynchrony*. There is no bound on message delivery delay or server execution speed.

### 2.1 Operations Implemented by COCA

COCA supports one operation (Update) to create, update, and invalidate certificates that specify bindings; a second operation (Query) retrieves certificates specifying those bindings. A client invokes an operation by issuing a *request* and then awaiting a *response*. COCA expects each request to contain a nonce. Responses from COCA are digitally signed using a COCA service key and include the client's request, hence the nonce,[1] thereby enabling a client to check whether a given response was produced by COCA for that client's request.

A request is considered *accepted* by COCA once any correct COCA server receives the request or participates in processing the request; and a request is considered *completed* once some correct server has constructed the response. It might, at first, seem more natural to deem a request "completed" only when the client receives a response. However, such a definition would make a client action (receipt of a response) necessary for a request to be considered completed. In the absence of assumptions about clients, it then becomes problematic for COCA to implement the following.

*Request Completion*. Every request accepted is eventually completed.

However, as will become clear, a correct client making a request will eventually receive a response from COCA.

Each COCA certificate $^3$ is a digitally signed attestation that specifies a binding between some name *cid* and some public key or other attributes *pubK*. In addition, each certificate $^3$ also contains a unique serial number ¾ ($^3$ ) assigned by COCA. The following semantics of COCA's Update and Query give meaning to the natural ordering on these serial numbers, namely—that a certificate for *cid* invalidates certificates for *cid* having lower serial numbers.

*Update*. Given a certificate $^3$ for a name *cid* and given a new binding *pubK*$^0$ for *cid*, an Update request returns an acknowledgment after COCA has created a certificate $^3$ $^0$ for *cid* such that $^3$ $^0$ binds *pubK*$^0$ to *cid* and ¾ ($^3$ ) < ¾ ($^3$ $^0$) holds.

*Query*. Given a name *cid*, a Query request *Q* returns a certificate $^3$ for *cid* such that:

(i) for any certificate $^3$ $^0$ for name *cid* created by an Update request that com-pleted before *Q* was accepted, ¾ ($^3$ $^0$) · ¾ ($^3$ ) holds.

By assuming an initial default binding for every possible name, the operation to create a first binding for a given name can be implemented by Query (to retrieve the certificate for the default binding) followed by Update. And an operation to revoke a certificate for *cid* is easily built from an Update specifying a new binding for *cid*.

Update creates and invalidates certificates, so its invocation should probably be restricted to certain clients. Consequently, COCA allows an authorization policy to be defined for Update. In principle, a CA could always process a Query, because Query does not affect any binding. In practice, that policy would create a vulnerability to denial-of-service attacks, so COCA adopts a more conservative approach discussed in Section 4.

The semantics of Update associates larger serial numbers with newer certificates and, in the absence of concurrent execution, a Query for *cid* returns the certificate whose serial number is the largest of all certificates for *cid*. Certificate serial numbers are actually consistent only with a *service-centric* causality relation: the transitive closure of relation !, where $^3$ !$^3$ $^0$ holds if and only if $^3$ $^0$ is created by an Update having $^3$ as input. Two Update requests *U* and *U*$^0$ submitted, for example, by the same client, serially, and where both input the same certificate, are not ordered by the relation. So, our semantics for Update allows *U* to create a certificate $^3$ , *U*$^0$ to create a certificate $^3$ $^0$, and ¾ ($^3$ $^0$) < ¾ ($^3$ ) to hold. This is consistent with the service-centric causality relation but the op-posite of what is required for serial numbers consistent with Lamport's [1978] more useful potential causality relation (because execution of

*U* is potentially causal for execution of $U^0$).

COCA is forced to employ the service-centric causality relation because COCA has no way to obtain information it can trust about causality involving operations it does not itself implement. Clients would have to provide COCA with that information, and compromised clients might provide bogus information.

Update and Query are not indivisible and (as shown in Section 3) are not easily made so: COCA's Update involves separate actions for the invalidation and for the creation of certificates. In implementing Update, we contemplated either possible ordering for these actions: execute invalidation first, and there is a period when no certificate is valid; execute invalidation last, and there is a period when multiple certificates are valid.

We wanted Query always to return a certificate, so avoiding periods with no valid certificate for a given name would have meant synchronizing Query with concurrent Update requests. We rejected this because the synchronization creates an execution-time cost and introduces a vulnerability to denial-of-service attacks—repeated requests by an attacker for one operation could now block requests for another operation. Our solution is to have Update create the new certificate before invalidating the old one, but it too is not without unpleasant consequences. Both of the following cannot now hold.

1. A certificate for *cid* is valid if and only if it is the certificate for *cid* with the largest serial number.
2. Query always returns a valid certificate.

COCA clients therefore must accommodate our more complicated semantics for Query and program their own synchronization.

## 2.2 Bounding the Window of Vulnerability

The duration of COCA's window of vulnerability cannot be characterized in terms of real time due to our Asynchrony assumption, so its duration is defined in terms of events marking the completion of *proactive recovery protocols* that are executed periodically to:

—reload the code (thereby eliminating Trojan horses);

—reconstitute the state of each COCA server (which might have been corrupted during the previous window of vulnerability); and

—make obsolete any confidential information an attacker might have obtained by compromising servers.

Each window of vulnerability at a COCA server begins when that server starts executing the proactive recovery protocols and terminates when that server has again started and finished those protocols. Thus every execution of the proactive recovery protocols is part of two successive windows of vulnerability. COCA is agnostic about when the proactive recovery protocols start. Currently, each COCA server attempts to run these protocols after a specified interval has elapsed on its local clock but (to avoid denial-of-service attacks) a server will refuse to participate in the protocols unless enough time has passed on its clock since they last executed.

In theory, using protocol events to delimit the window of vulnerability affords attackers leverage. Denial-of-service attacks that slow servers and/or increase message delivery delays expand the real-time duration for the window of vulnerability, creating a longer period during which attackers can try to compromise more than *t* servers. But, in practice, we expect assumptions about timing can be made for those portions of the system that have not been compromised.[2] Given such information about correct server execution speeds and message-delivery delays, real-time bounds on the window of vulnerability can be computed.

## Limiting the Utility of Compromised Keys

*Server Keys*. Each COCA server maintains a private/public key pair, and the public key is known by all COCA servers. These public keys allow servers to authenticate the senders of messages they exchange with other servers.

In the absence of tamper-proof coprocessors, server keys must be refreshed as part of proactive recovery. One simple approach has trusted administrators for each server invent and propagate new public keys through secure channels implemented by having an *administrative* public/private key pair. The administrative public key is known to other administrators (and all servers); the administrative private key, kept offline most of the time as a defense against online attacks, is used to sign notification messages for the new public server public key. Other rekeying schemes are discussed in Canetti and Herzberg [1994].

Public keys of COCA servers are not given to COCA clients so that clients need not be informed of changed server keys, which is attractive in a system with a large number of clients and where server keys are periodically refreshed.

*Service Key*. There is one service private/public key pair. It is used for signing responses and certificates. All clients and servers know the service public key.

[2]A server that violates these stronger execution timing assumptions might be considered compromised, for example.

The service private key is held by no COCA server. Instead, different shares of the key are stored on each of the servers, and threshold cryptography [Desmedt and Frankel 1990, 1992; Desmedt 1994, 1998; Frankel and Yung 1998] is used to construct signatures on responses and certificates. To sign a message:

1. each COCA server generates a *partial signature* from the message and that server's share of the service private key;
2. some COCA server combines these partial signatures and obtains the signed message.[3]

With ($n$, $t$ C 1) threshold cryptography, $t$ C 1 or more partial signatures are needed in order to generate a signature. An adversary must therefore com-promise $t$ C 1 servers in order to forge COCA signatures.

*Proactive Secret-Sharing*. A mobile adversary might compromise $t$ C 1 servers over a period of time and, in so doing, collect the $t$ C 1 shares of the service private key. Consequently, COCA employs a proactive secret-sharing protocol to refresh these shares, periodically generating a new set of shares for the service private key and deleting the old set. New shares cannot be combined with old shares to construct signatures. So periodic execution of this proactive secret-sharing protocol ensures that a mobile adversary can forge COCA sig-natures only by compromising $t$ C 1 servers in the interval between protocol executions.

The proactive secret-sharing protocol that COCA employs makes no syn-chrony assumptions (which would be incompatible with the asynchrony as-sumption of Section 2), unlike prior work (e.g., Jarecki [1995], Herzberg et al. [1995, 1997], and Frankel et al. [1997a,b]); details are discussed in Zhou et al. [2002], and Zhou [2001]. For the discussion in this article, it suffices to regard the protocols simply as services that COCA invokes.

*Server Code and State Recovery*. Part of proactive recovery should include refreshing the states and reloading the code at COCA servers. The state of a COCA server involves a set of certificates. In theory, this state could be re-freshed by performing a Query request for each name that could appear in a certificate, but the cost of such an enumeration would be prohibitive. So in-stead, during proactive recovery, a list with the name and serial number for every valid certificate stored by each COCA server is sent to every other server. Upon receiving this list, a server retrieves any certificates that appear to be missing. Certificates stored by COCA servers are signed (by COCA), so each certificate can be checked to make sure it is not bogus. The certificate serial numbers enable servers to determine which of their certificates have been in-validated (because a certificate for that same name but with a higher serial number exists).

[3]Having a client combine the partial signatures instead of having COCA do it introduces a vulner-ability to denial-of-service attacks. Clients, lacking COCA server public keys, do not have a way to authenticate the origins of messages conveying the partial signatures. Therefore a client could be bombarded with bogus partial signatures, and only by actually trying to combine these fragments (an expensive enterprise) could the bona fide partial signatures be identified.

Server code should be reloaded from some read-only medium or other trusted source by proactive recovery in order to eliminate Trojan horses installed by attackers during the previous window of vulnerability. This functionality is not currently implemented in our prototype, however, since defending against such attacks is not the focus of our research; see Castro [2000] for an in-depth discussion of the issues.

There is one non-obvious point of interaction between proactive recovery and request processing. To satisfy Request Completion, an accepted request that has not been completed when a window of vulnerability ends must become an accepted request in the next window of vulnerability. Therefore such a request must be propagated to other servers as part of proactive recovery. So each correct server, when executing the proactive recovery protocol, resubmits to all servers any request that is then in progress and awaits acknowledgments from at least $t$ C 1 servers. Some server that is correct in this next window of vulnerability necessarily receives that request, and that means this accepted request in the previous window of vulnerability also becomes an accepted request in the new window of vulnerability.

To avoid a spate of new requests from delaying termination of proactive recovery (a potential denial-of-service attack), COCA servers could ignore such new requests.[4] In those rare cases where a restarted request has not finished before a new proactive recovery is started, COCA could delay proactive re-covery until after the processing of restarted requests has been completed. In practice, windows of vulnerability will tend to be long (viz. days) relative to the time (5 seconds or less) required for processing a Query or Update request. It is thus extremely unlikely that a request restarted in a subsequent window of vulnerability would not be completed before proactive recovery is again commenced.

## 3. PROTOCOLS

In COCA, every client request is processed by multiple servers and every certificate is replicated on multiple servers. The replication is managed as a dissemination Byzantine quorum system [Malkhi and Reiter 1998a], which is feasible because we have assumed $3t$ C 1 · $n$ holds. So servers are organized by COCA into sets, called *quorums*, satisfying[5] the following.

*Quorum Intersection*. The intersection of any two quorums contains at least one correct server.

*Quorum Availability*. A quorum comprising only correct servers always exists.

And every client request is processed by all correct servers in some quorum.

[4]The time to execute proactive recovery tends to be short, and ignoring (a finite number of) messages is permitted by the Fair Links assumption.

5 Provided there are 3*t* C 1 servers and at most *t* of those servers may be compromised, the quorum system constitutes a dissemination Byzantine quorum system. For simplicity, we assume *n* D 3*t* C 1 holds; the protocols are easily extended to cases where *n* > 3*t* C 1 holds.

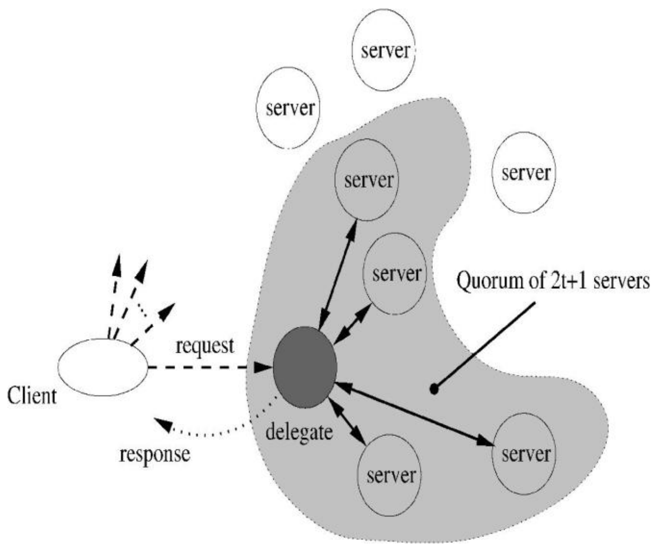


Fig. 1. Overview of client request processing.

Detailed protocols for Query and Update appear as an Appendix; in this section, we explain the main ideas behind the design of these protocols. Technical challenges the protocols must address include the following.

—Because requests are processed by a quorum of servers but not necessarily by all correct COCA servers, different correct servers might process different Update requests. Consequently, different certificates for a given name *cid* are stored by correct servers. Certificate serial numbers provide a solution to the problem of determining which of those certificates is the one to use.

—Because clients do not know COCA server public keys, a client making a request cannot authenticate messages from a COCA server and, therefore, cannot determine whether a quorum of servers has processed that request. The solution is for some COCA servers to become *delegates* for each request. A delegate presides over the processing of a client request and, being a COCA server, can authenticate server messages and assemble the needed partial signatures from other COCA servers. A client request is handled by *t* C 1 delegates to ensure that at least one of these delegates is correct.

—Because communication is done using fair links, retransmission of messages may be necessary.

Figure 1 gives a high-level view of how COCA operates by depicting one of the *t* C 1 delegates and the quorum of servers working with that delegate to handle a client request. The figure shows a client making its request by sending a signed message to *t* C 1 COCA servers. Each server that receives this message assumes the role of delegate for the request. A delegate engages a quorum of servers to handle the request (by sending that request to all COCA servers) and constructs a response to the request based on the responses received from that quorum. The delegate then causes this response to be signed by the service;
this involves running a threshold signature protocol in cooperation with *t* other servers. Once signed, the response is sent by the delegate to the client. Upon receipt, the client checks that the response is correctly signed by the service and contains the client's original request; if it isn't or if no response has been received within a specified period of time, then the client simply again sends the original request to *t* C 1 servers.

## Protocol Details

*Certificate Serial Numbers.* The serial number ¾ (³ ) for a COCA certificate ³ is implemented as a pair h*v*(³ ), *h*(*R*₃ )i, where *v*(³ ) is a *version number* and *h*(*R*₃ ) is a collision-resistant hash of the Update request *R*₃ that led to creation of ³ . Version numbers encode the service-centric causality relation as follows.

—The first certificate created to specify a binding for a name *cid* is assigned version number 0.

—A certificate ³ $^0$ produced by an Update given certificate ³ is assigned version number *v*(³ $^0$) D *v*(³ ) C 1.

Because different requests have different collision-resistant hashes, certificates created by different requests have different serial numbers. The usual lexicographic ordering on serial numbers yields the total ordering on serial numbers we seek—an ordering consistent with the transitive closure of the ! relation.

Note that, even with serial numbers on certificates, the same new certificate will be created by COCA if an Update request is resubmitted, and Update re-quests are thus idempotent. This is because the serial number of a certificate is entirely determined by the arguments in the request that creates the certificate.

*Determining a Response for* Query. It suffices to consider an abstract description of COCA's Update and Query protocols in order to characterize responses satisfying Parts (i) and (ii) in the specification for Query. The actual protocols refine this abstract description.

COCA Update requests are processed by correct servers in some quorum and not necessarily by all correct COCA servers. Consequently, a correct COCA server *p* can be ignorant of certificates having larger serial numbers than *p* stores for a name *cid*. Part (ii) in the specification for Query implies that all completed Update requests (hence, all certificates) must be taken into account in determining the response to a Query request *Q*. Therefore, a quorum of servers must be engaged in processing *Q*. Responses from a quorum *Q* of servers is guaranteed if all COCA servers are contacted. Provided each server in *Q* responds with the certificate (signed by COCA) it stores having the largest serial number among all certificates (for *cid*) known to the server, then the certificate ³ having the largest serial number among the correctly signed certificates received in the responses from *Q* can serve as the response to *Q*. That is, ³ will satisfy Parts (i) and (ii) in the specification for Query, as we now show.

We first show that any certificate ³ obtained by refining the protocol outlined above satisfies Part (i). Part (i) stipulates that a certificate returned for Query is created by an accepted Update; it is satisfied by ³ if each certificate is signed by COCA only after the Update request creating that certificate has been accepted. This is because the (*n*, *t* C 1) threshold cryptography being employed for digital signatures requires cooperation (collusion) by more than *t* servers in order to sign a certificate. Given our assumption of at most *t* compromised servers, we conclude that there are not enough compromised servers to create bogus signed certificates. Therefore, when a certificate is signed, a correct server must have participated in processing the request that created the certificate; the request creating the certificate had to have been accepted.

Part (ii) of the specification for Query requires that, for any Update request *U* naming *cid* and completed before *Q* is accepted, ¾ (³ $^0$) · ¾ (³ ) must hold where ³ $^0$ is the certificate created by *U*. This holds for implementations that refine the abstract description given above due to Quorum Intersection, because some correct server *p* in *Q* must also be in the quorum that processed *U*. Let certificate ³$_p$ be *p*'s response for *Q*. Server *p* always chooses the certificate for *cid* with the largest serial number, so ¾ (³ $^0$) · ¾ (³$_p$) holds. And because ³ is the certificate that has the largest serial number among those from all servers in *Q*, ¾ (³$_p$) · ¾ (³ ) holds. Therefore ¾ (³ $^0$) · ¾ (³ ) holds.

*The Role of Delegates.* Every request is processed by all correct servers in some quorum; the client must be notified once that has occurred. Direct notification by servers in the quorum is not possible because clients do not know the public keys for COCA servers and, therefore, have no way to authenticate messages from those servers. So, instead, a COCA server is employed to detect the completion of request processing and then to notify the client, as follows.

A delegate for a request *R* is a COCA server that causes *R* to be processed by correct COCA servers in some quorum and then sends a response (signed by COCA) back to the initiating client. The processing needed to construct the response depends on the type of request being processed.

—To process a Query request *Q* for name *cid*, the delegate obtains certificates from a quorum of servers, picks the certificate ³ having the largest serial num-ber, and uses the threshold signature protocol to produce a signed response containing ³ :

1. Delegate forwards *Q* to all COCA servers.
2. Delegate awaits certificates for *cid* from a quorum of COCA servers.
3. Delegate picks the certificate ³ having the largest serial number of those received in Step 2.
4. Delegate invokes COCA's threshold signature protocol to sign a response containing ³ ; that response is sent to the client.

—To process an Update request *U* for name *cid*, the delegate constructs the certificate ³ for the given new binding (using the threshold signature protocol to have COCA digitally sign it) and then sends ³ to all COCA servers. A server
*p* replaces the certificate ³$_p^{cid}$ for *cid* that it stores by ³ if and only if the serial number in ³ is larger than the serial number in ³$_p^{cid}$.

1. Delegate constructs a new certificate ³ for *cid*, using the threshold signature protocol to sign the certificate.
2. Delegate sends ³ to every COCA server.
3. Every server, upon receipt, replaces the certificate for *cid* it had been storing if the serial number in ³ is larger. The server then sends an acknowledgment to the delegate.
4. Delegate awaits these acknowledgments from a quorum of COCA servers.
5. Delegate invokes COCA's threshold signature protocol to sign a response; that response is sent to the client.

Quorum Availability ensures that a quorum of servers is always available, so Step 2 in Query and Step 4 in Update are guaranteed to terminate. Since at least *t* C 1 of COCA's 3*t* C 1 servers are correct, compromised servers cannot prevent a delegate from using (*n*, *t* C 1) threshold cryptography in constructing the COCA signature for a certificate or a response. Thus, Step 4 in Query and Steps 1 and 5 in Update, which involve contacting all COCA servers, cannot be disrupted by compromised servers.

A compromised delegate might not follow the protocol just outlined for pro-cessing Query and Update requests. COCA ensures that such behavior does not disrupt the service by enlisting *t* C 1 delegates (instead of just one) for each request. At least one of the *t* C 1 delegates must be correct, and this delegate can be expected to follow the Query and Update protocols. So we stipulate that a (correct) client making a request to COCA submits that request to *t* C 1 COCA servers; each server then serves as a delegate for processing that request.[6]

With *t* C 1 delegates, a client might receive multiple responses to each request and each request might be processed repeatedly by some COCA servers. The du-plicate responses are not difficult for clients to deal with: a response is discarded if it is received by a client not waiting for a request to be processed. That each request might be processed repeatedly by some COCA servers is not a problem either, because COCA's Query and Update implementations are idempotent.

But a compromised client might not follow the protocol and thus might not submit its request to *t* C 1 delegates. A problem then occurs if the delegates receiving a request *R* execute the first step of Query or Update processing and then halt. Correct COCA servers now participated in the processing of $R$, so (by definition) $R$ is accepted. Yet no (correct) delegate is responsible for $R$. Request $R$ is never completed, and Request Completion is violated. The defense is straightforward.

—Messages related to the processing of a client request $R$ contain $R$.

—Whenever a COCA server receives a message related to processing a client request $R$, that server becomes a delegate for $R$ if it is not already serving as one.

The existence of a correct delegate is now guaranteed for every request that is accepted.

*Self-Verifying Messages.* Compromised delegates might also attempt to produce an incorrect (but correctly signed) response to a client by sending erroneous messages to COCA servers. For example, in processing a Query request, a compromised delegate might construct a response containing a bogus or in-validated certificate and try to get other servers to sign that; in processing an Update request, a compromised delegate might create a fictitious binding and try to get other servers to sign that; or when processing an Update re-quest, a compromised delegate might not disseminate the updated certificate to a quorum (causing the response to a later Query to contain an invalidated certificate).

COCA's defense against erroneous messages from compromised servers is a form of monitoring and detection that we call *self-verifying messages*.[7] A self-verifying message comprises:

—information the sender intends to convey, and

—evidence enabling the receiver to verify—without trusting the sender—that the information being conveyed by the message is consistent with some given protocol and also is not a replay.

In COCA, every message a delegate sends on behalf of a request contains a transcript of relevant messages previously sent and received in processing that request (including the original client request). Because messages contained in the transcript are signed by their senders, a compromised delegate cannot forge the transcript. And, because the members of the quorum participating in the protocol are known to all, the receiver of such a self-verifying message can independently establish whether messages sent by a delegate are consistent with the protocol and the messages received.[8]

*Communicating Using Fair Links.* The Fair Links assumption means that not all messages sent are delivered. To implement reliable communication in this environment, it suffices for a sender to resend each message until a signed acknowledgment is received from the intended recipient. In turn, the recipient returns a signed acknowledgment for every message it receives (including duplicates, since the previous acknowledgments could have been lost). If both the sender and the receiver are correct then (due to Fair Links) this protocol ensures that the receiver eventually receives the message, the sender eventually receives an acknowledgment from the receiver, and the sender exits the protocol.

Each protocol in COCA is structured as a series of multicasts, with information piggybacked on the acknowledgments. A client starts by doing a multicast to $t$ C 1 delegates; the signed response from a single delegate can be considered the acknowledgment part of that multicast. A delegate then inter-acts with COCA servers by performing multicasts and awaiting responses from

[7]Similar schemes can be found in Kihlstrom et al. [1997], Castro and Liskov [1999], Baldoni et al. [2000], and Doudou et al. [2000].

8 Gong and Syverson [1998] introduce the notion of a *fail-stop protocol*, which is a protocol that halts in response to certain attacks. One class of attacks is thus transformed into another, more benign, class. Our self-verifying

messages can be seen as an instance of this approach, transforming certain Byzantine failures to more-benign failures.

servers. For the threshold signature protocol, $t$ C 1 correct responses suffice; for retrieving and for updating certificates, responses from a quorum of servers are needed. Thus, with at least $2t$ C 1 correct servers, COCA's multicasts al-ways terminate due to Quorum Availability since a delegate is now guaranteed to receive enough acknowledgments at every step and, therefore, eventually that delegate will stop retransmitting messages.

## 4. DEFENSE AGAINST DENIAL-OF-SERVICE ATTACKS

A large number of successful denial-of-service attacks work by exploiting an imbalance between the resources an attacker must expend to submit a request and the resources the service must expend to satisfy that request, as has been noted, for example, in Juels and Brainard [1999], and Meadows [1999, 2001]. If making a request is cheap but processing one is not, then attackers have a cost-effective way to disrupt a service: submit bogus requests to saturate server resources. A service, like COCA, where request processing involves expensive cryptographic operations and multiple rounds of communication is especially susceptible to such resource-clogging attacks.

COCA implements several classic defenses to blunt resource-clogging denial-of-service attacks.

1. An authorization mechanism identifies requests on which resources should not be expended.
2. Requests are grouped into classes, and resources are scheduled in a manner that prevents demands by one class from affecting requests in another class.
3. The results of expensive cryptographic operations are cached, and attackers cannot destroy the locality that makes this cache effective.

The details for COCA's realizations of these defences constitute the bulk of this section.

Note that our Fair Links and Asynchrony assumptions are an important defense against denial-of-service attacks, too. An attacker stealing network bandwidth or cycles from processors that run COCA servers is not violating assumptions needed for COCA's algorithms to work. Such a "weak assumptions" defense is not without a price, however. Implementing real-time service guarantees on request processing requires a system model with stronger assumptions than we are making. Consequently, COCA can guarantee only that requests it receives are processed eventually. Those who equate availability with real-time guarantees (e.g., Gligor [1984], Yu and Gligor [1990], and Millen [1992, 1995]) would not be satisfied by an eventuality guarantee. But a sys-tem whose correctness depends only on "weak assumptions" is not precluded from satisfying real-time guarantees when the environment satisfies stronger assumptions, and COCA does just that.

Finally, COCA employs connectionless protocols for communication with clients and servers, so COCA is not susceptible to connection-depletion attacks such as the well-known TCP SYN flooding attack [Schuba et al. 1997]. But the proactive secret-sharing protocol in the current COCA implementation does use SSL (secure socket layer) [Freier et al. 1996] and is, therefore, subject to certain denial-of-service attacks. This vulnerability could be eliminated by restricting the rate of SSL connection requests, reprogramming the proactive secret-sharing protocol, or adopting the mechanisms described in Juels and Brainard [1999].

## 5. PERFORMANCE OF COCA

Our COCA prototype is approximately 35 K lines of new C source; it employs threshold and proactive threshold RSA schemes (with 1,024-bit RSA keys), constructed using the protocol described in Zhou [2001] from building blocks given in Rabin [1998].[12] We implemented the protocols in OpenSSL [OpenSSL Project]. COCA certificates have the same syntax as X.509 [CCITT 1988]

[10]In a system with a million clients, this client cache would be roughly 5 gigabytes because approx-imately 5 K bytes are needed to store a client's last request and COCA's response.
[11]The one-way hash function involves expensive modular exponentiation and is needed to imple-ment verifiable secret-sharing [Feldman 1987].
12 The protocols [Zhou 2001] we use employ replication of shares and subshares in achieving fault tolerance rather than the backup scheme used in Rabin [1998].

certificates, with a COCA certificate's serial number embedded in the X.509 serial number.[13]

Much of the cost and complexity of COCA's protocols is concerned with tolerating failures and defending against attacks, even though failures and attacks are infrequent today. We normally expect the following.

N1: Servers will satisfy stronger assumptions about execution speed. N2: Messages sent will be delivered in a timely way.

Our COCA prototype is optimized for these normal circumstances. Wherever possible, redundant processing is delayed until there is evidence that assumptions N1 and N2 no longer hold.

In particular, our prototype delays when COCA servers start serving as the additional delegates for client requests already in progress. This reduces the number of delegates when N1 and N2 hold, hence it reduces the cost of re-quest processing in normal circumstances. The refinements to the protocols of Section 3 are as follows.

—A client sends its request only to a single delegate at first. If this delegate does not respond within some timeout period, then the client sends its request to another *t* delegates, as required by the protocols in Section 3.

—A server that receives a message in connection with processing some client request *R* and that is not already serving as a delegate for *R* does not become a delegate until some timeout period has elapsed.

—A delegate *p* sends a response to all COCA servers, in addition to sending the response to the client initiating the request, after the request has been processed. After receiving such a response, a server that is not yet a delegate for this request will not become one in the future; a server that is serving as a delegate aborts that activity.

—A cached response is forwarded to a server *q* whenever *q* instructs *p* to par-ticipate in the processing of a request that has already been processed. Upon receiving the forwarded response, *q* immediately terminates serving as a delegate for that request.

Also, the threshold signature protocol COCA uses is designed to give better performance when N1 and N2 hold.

## 5.1 Local Area Network Deployment

The experiments reported in this subsection all involved a COCA prototype comprising four servers (i.e., $n$ D 4 and $t$ D 1) communicating using a 100-Mbps Ethernet. The servers were Sun E420R Sparc systems running Solaris 2.6, each

[13]Although syntactically compatible with X.509 certificates, COCA certificates are not interchange-able with the X.509 certificates in use by public key infrastructures today. First, COCA imposes an interpretation on the serial numbers embedded in certificates: a COCA certificate with a higher serial number

invalidates one with a lower serial number for the same client. Second, COCA, because it supports Query, has no need to and therefore does not provide the CRLs (certification revocation lists) usually associated with public key infrastructures that support X.509 certificates.

| | Query (%) | Update (%) | PSS (%) |
|---|---|---|---|
| Partial Signature | 64 | 73 | |
| Message Signing | 24 | 19 | 22 |
| One-Way Function | | | 51 |
| SSL | | | 10 |
| Idle | 7 | 2 | 15 |
| Other | 5 | 6 | 2 |

Table II. Cost Breakdown for Query, Update, and Proactive Secret-Sharing (PSS) in LAN Deployment

with four 450-MHz processors. The round trip delay for a UDP packet between any two servers on this Ethernet is usually under 300 microseconds.

Table I gives times for COCA operations executing in isolation when assumptions N1 and N2 hold. We report the delay for Query, for Update, and for a round of proactive secret-sharing.[14] The reported sample means and sample standard deviations are based on 100 executions. All samples were within 5% of the mean.

To better understand the origin of these delays, we report in Table II the (percentage) contribution that can be attributed to certain CPU-intensive cryptographic operations. For Query and Update, we measured the time spent generating partial signatures and signing messages. For proactive secret sharing, we measured the delay associated with the one-way function, with message signing, and with computation involved in establishing an SSL connection to transmit confidential information between servers. Notice that improved hard-ware for performing cryptographic operations could have a considerable im-pact. Idle time, because servers sometimes wait for each other,

Table I. Execution Time in a LAN When N1 and N2 Hold

| COCA Operation | Mean (msec) | Std dev. (msec) |
|---|---|---|
| Query | 629 | 16.7 |
| Update | 1109 | 9.0 |
| PSS | 1990 | 54.6 |

is also listed in Table II. Only 2 to 6% of the total execution time is unaccounted. That time is being used for signature verification, message marshaling and unmarshaling, and task management.

To evaluate the effectiveness of the optimizations outlined above for when assumptions N1 and N2 hold, Figure 2 compares performance with and without the optimizations. The results summarize 100 executions; very small sample standard deviations are observed here. The optimizations thus can be seen to be effective.

## 5.2 Internet Deployment

Communication delays in the Internet are higher than in a local area network; the variance of these delays is also higher. To understand the extent, if any, to

[14]Time spent checking certificates and performing other state recovery at each server is not included in these delays.

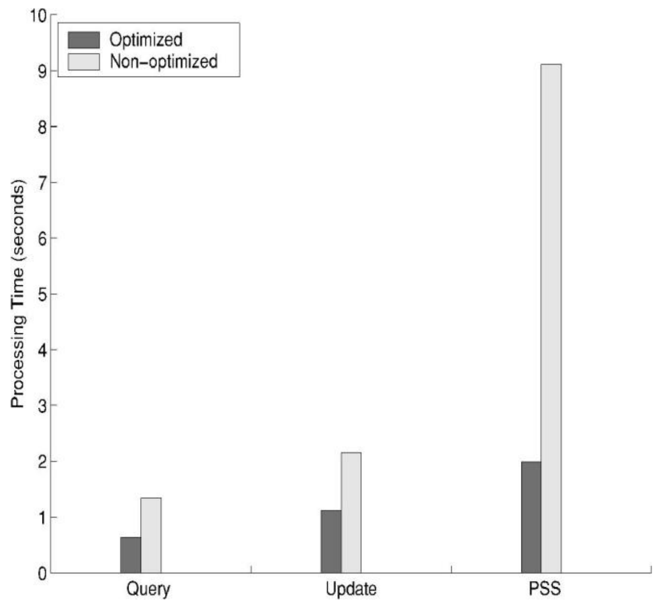


Fig. 2. Effectiveness of optimization in Query, Update, and proactive secret-sharing when assump-tions N1 and N2 hold.

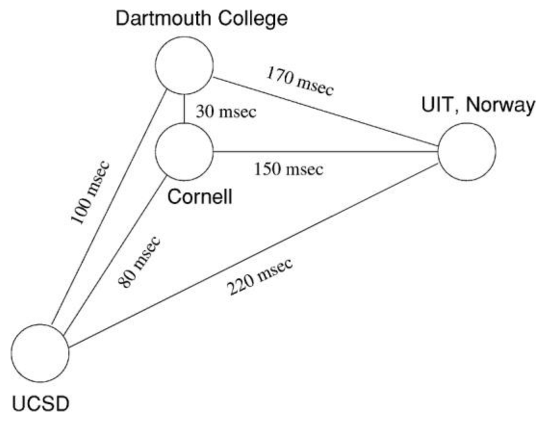
which this affects performance, we deployed four COCA servers as follows.

—University of TromsÁ (UIT), TromsÁ, Norway (300 MHz, Pentium II).
—University of California at San Diego (UCSD), San Diego, CA (266 MHz, Pentium II).
—Cornell University, Ithaca, NY (550 MHz, Pentium III).
—Dartmouth College, Hanover, NH (450 MHz, Pentium II).

All ran Linux.[15] Figure 3 depicts the average message delivery delay (measured using ping) between these servers. Delivery delays on the Internet vary considerably [Labovitz et al. 1997] but the values observed during the experiments we report did not differ significantly from those in Figure 3.

Table III gives measurements for the Cornell host in our four-site Internet deployment. In comparing Tables I and III, we see the impact of the Internet's longer communication delays (which also lead to longer server idle times). The sample standard deviation is also higher for the Internet deployment, due to higher load variations on servers and due to the higher variance of delivery delays on the Internet; all samples are located within 25% of the mean. See Table IV for a breakdown of delays (analogous to Table II) for our Internet deployment of COCA.

[15]Beggars can't be choosers. For making measurements, we would have preferred having the same hardware at every site, although we have no reason to believe that our conclusions are affected by the modest differences in processor speeds. For a production COCA deployment, we would recommend having different hardware and different operating systems at each site so that common-mode vulnerabilities are reduced.



## Threshold Signature Protocol

The following describes the threshold signature protocol[22] *threshold sign*(*m*, *E*), where *m* is the message to be signed and *E* is the evidence used in self-verifying

[22]Although this protocol is appropriate for schemes such as threshold RSA, the protocol might not be applicable to other threshold signature schemes, such as those based on discrete logarithms (e.g., Cerecedo et al. [1993]

and Harn [1994]). Those schemes may require an agreed-upon random number in generating partial signatures. Such schemes can be implemented by adding a new first messages to convince receivers to generate partial signatures for *m*. As detailed in Appendices A.3 and A.4, different evidence is used in the protocols for Query and Update.

1. Server *p* sends to each server *q* a sign request message with message *m* to be signed and evidence *E*.

2. Each server *q*, upon receiving a sign request message (1), verifies evidence *E* with respect to *m*. If *E* is valid, then *q* generates a partial signature using its share $s_q$ and sends the partial signature back to *p*.

   [$q$ ! $p$ : hsign response, $q, p, m, PS(m, s_q)$i$_q$ ]

3. Server *p* periodically repeats Step 1 until it receives partial signatures from a quorum of servers[23] (which includes a partial signature from *p* itself). Server *p* then selects $t$ C 1 partial signatures to construct signature h$m$i$_k$ . If the resulting signature is invalid (which would happen if compromised servers submit erroneous partial signatures), then *p* tries another combination of $t$ C 1 signatures.[24] This process continues until the correct signature h$m$i$_k$ is obtained.

## REFERENCES

[1] ADAMS, C., SYLVESTER, P., ZOLORAREV, M., AND ZUCCHERATO, R. 2001. Data validation and certification server protocols. Request for Comments 3029.

[2]BALDONI, R., HELARY, J.-M., AND RAYNAL, M. 2000. From crash fault-tolerance to arbitrary-fault tolerance: Towards a modular approach. In *Proceedings of the International Conference on De-pendable Systems and Networks*, IEEE Computer Society Technical Committee on Fault-Tolerant Computing, IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance, IEEE Computer Society, New York, 273–282.

[3] BARAK, B., HERZBERG, A., NAOR, D., AND SHAI, E. 1999. The proactive security toolkit and applica-tions. In *Proceedings of the Sixth ACM Conference on Computer and Communications Security (CCS'99)*, ACM SIGSAC, ACM, Kent Ridge Digital Labs, Singapore, 18–27.

[4] BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC'88*, ACM, Chicago, 1–10.

[5]BURMESTER, M., DESMEDT, Y., AND KABATIANSKI, G. 1998. Trust and security: A new look at the Byzantine generals problem. In *Network Threats, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 38, R. N. Wright and P. G. Neumann, Eds., American Mathe-matical Society, Providence, R.I., 75–83.

[5] BYRD, G. T., GONG, F., SARGOR, C., AND SMITH, T. J. 2001. Yalta: A secure collaborative space for dynamic coalitions. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, IEEE, United States Military Academy, West Point, 30–37.

[6] CACHIN, C. 2001. Distributing trust on the Internet. In *Proceedings of International Conference on Dependable Systems and Networks (DSN-2001)*, IEEE Computer Society Technical Committee on Fault-Tolerant Computing, IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance, IEEE Computer Society, Goteborg,¨ Sweden, 183–192.

[7] CACHIN, C. AND PORITZ, J. A. 2001. Hydra: Secure replication on the Internet. Techn. Rep. RZ 3393, IBM Research. December.

[8] CANETTI, R. AND HERZBERG, A. 1994.

Maintaining security in the presence of transient faults. In *Advances in Cryptology—Crypto'94, the Fourteenth Annual International Cryptology Conference, Proceedings, Lecture Notes in Computer Science*, vol. 839, Y. Desmedt, Ed., Springer-Verlag, Berlin, 425–438.

[9] CASTRO, M. 2000. Practical Byzantine fault tolerance. PhD. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass.

[10]CASTRO, M. AND LISKOV, B. 1999. Practical Byzantine fault tolerance. In *Proceedings of the third USENIX Symposium on Operating System Design and Implementation (OSDI'99)*, USENIX As-sociation, IEEE TCOS, and ACM SIGOPS, New Orleans, 173–186.

[11] CASTRO, M. AND LISKOV, B. 2000. Proactive recovery in a Byzantine-fault-tolerant system. In *Pro-ceedings of the Fourth USENIX Symposium on Operating System Design and Implementation (OSDI'00)*, USENIX Association, IEEE TCOS, and ACM SIGOPS, San Diego, 273–287.

[12]CCITT. 1988. Recommendation X.509: The directory-authentication framework.

[13]CERECEDO, M., MATSUMOTO, T., AND IMAI, H. 1993. Efficient and secure multiparty generation of digital signatures based on discrete logarithms. *IEICE Trans. Fund. Electro. Inf. Commun. Eng. E76-A*, 4 (April), 532–545.

[14]CERTCO, INC. Available at http://www.certco.com.

[15]CHAUM, D. 1983. Blind signatures for untraceable payments. In *Advances in Cryptology— Crypto'82, A Workshop on the Theory and Application of Cryptography, Proceedings*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Plenum, New York, 199–203.

[16]CHAUM, D. 1985. Security without identification: Transaction systems to make big brother obso-lete. *Commun. ACM 28*, 10 (October), 1030–1044.

[17]CHAUM, D. 1988. Blinding for unanticipated signatures. In *Advances in Cryptology— Eurocrypt'87, Workshop on the Theory and Application of Cryptographic Techniques, Proceedings, Lecture Notes in Computer Science*, vol. 304, D. Chaum and W. L. Price, Eds., Springer-Verlag, Berlin, 227–233.

[18]CHAUM, D., CREPEAU´, C., AND DAMGARD˚, I. 1988. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC'88*, ACM, Chicago, 11–19.

[19]DESMEDT, Y. 1994. Threshold cryptography. *European Trans. on Telecommun. 5*, 4 (July–August), 449–457.

[20]DESMEDT, Y. 1998. Some recent research aspects of threshold cryptography. In *Information Secu-rity, The First International Workshop, ISW'97, Proceedings, Lecture Notes in Computer Science*, vol. 1396, E. Okamoto, G. Davida, and M. Mambo, Eds., Springer-Verlag, Berlin, 158–173.

[21]DESMEDT, Y. AND FRANKEL, Y. 1990. Threshold cryptosystems. In *Advances in Cryptology— Crypto'89, the Ninth Annual International Cryptology Conference, Proceedings, Lecture Notes in Computer Science*, vol. 435., G. Brassard, Ed., Springer-Verlag, Berlin, 307–315.

[22]DESMEDT, Y. AND FRANKEL, Y. 1992. Shared generation of authenticators and signatures (Extended Abstract). In *Advances in Cryptology—Crypto'91, the Eleventh Annual International Cryptol-ogy Conference, Proceedings, Lecture Notes in Computer Science*, vol. 576, J. Feigenbaum, Ed., Springer-Verlag, Berlin, 457–469.

[23]DOUDOU, A., GARBINATO, B., AND GUERRAOUI, R. 2000. Modular abstractions for devising Byzantine-resilient state machine replication. In *Proceedings of the Nineteenth IEEE Symposium on Reliable Distributed Systems*., IEEE Computer Society, Nurnberg,¨

Germany, 144–153.

[24]DRAELOS, T., HAMILTON, V., AND ISTRAIL, G. 1998. Proactive DSA application and implementation. Tech. Rep. SAND—97-2939C; CONF-980554—, Sandia National Laboratories, Albuquerque, May 3.

[25]FELDMAN, P. 1987. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on the Foundations of Computer Science*, IEEE, New York, 427– 437.

[26]FISCHER, M. J., LYNCH, N. A., AND PETERSON, M. S. 1985. Impossibility of distributed consensus with one faulty processor. *J. ACM 32*, 2 (April), 374–382.

[27]FOX, B. AND LAMACCHIA, B. 1998. Certificate revocation: Mechanics and meaning. In *Financial Cryptography, the Second International Conference (FC'98), Proceedings*, *Lecture Notes in Com-puter Science*, vol. 1465, R. Hirschfeld, Ed., Springer-Verlag, Berlin, 158–164.

[28]FRANKEL, Y., GEMMEL, P., MACKENZIE, P., AND YUNG, M. 1997a. Optimal resilience proactive public-key cryptosystems. In *Proceedings of the 38th Symposium on Foundations of Computer Science*, IEEE, Miami Beach, 384–393.

[29]FRANKEL, Y., GEMMELL, P., MACKENZIE, P., AND YUNG, M. 1997b. Proactive RSA. In *Advances in Cryptology—Crypto'97, Proceedings*, *Lecture Notes in Computer Science*, vol. 1294, B. S. Kaliski, Jr., Ed., Springer-Verlag, Berlin, 440–454.

[30]FRANKEL, Y. AND YUNG, M. 1998. Distributed public key cryptosystem. In *Public Key Cryptography, the First International Workshop on Practice and Theory in Public Key Cryptography, PKC'98, Proceedings*, *Lecture Notes in Computer Science*, vol. 1560, H. Imai and Y. Zheng, Eds., Springer-Verlag, Berlin, 1–13.

[31]FREIER, A. O., KARLTON, P., AND KOCHER, P. C. 1996. The SSL protocol Version 3.0. Internet Draft. GARAY, J. A., GENNARO, R., JUTLA, C., AND RABIN, T. 2000. Secure distributed storage and retrieval. *Theor. Comput. Sci. 243*, 1–2 (July 28), 363–389.

[32]GASSER, M., GOLDSTEIN, A., KAUFMAN, C., AND LAMPSON, B. 1989. The digital distributed systems security architecture. In *Proceedings of the Twelfth National Computer Security Conference*, National Institute of Standards and Technology (NIST), National Computer Security Center (NCSC), National Institute of Standards and Technology (NIST), Baltimore, 305–319.

[33]GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. 1996a. Robust threshold DSS signatures. In *Advances in Cryptology—Eurocrypt'96, International Conference on the Theory and Application of Cryptographic Techniques, Proceedings*, *Lecture Notes in Computer Science*, vol. 1233, U. M. Maurer, Ed., Springer-Verlag, Berlin, 354–371.

[34]GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. 1996b. Robust and efficient sharing of RSA functions. In *Advances in Cryptology—Crypto'96, the Sixteenth Annual International Cryptology Conference, Proceedings*, *Lecture Notes in Computer Science*, vol. 1109, N. Koblitz, Ed., Springer-Verlag, Berlin, 157–172.

[35]GLIGOR, V. D. 1984. A note on denial-of-service in operating systems. *IEEE Trans. Softw. Eng. 10*, 3 (May), 320–324.

[36]GOLDREICH, O., MICALI, S., AND WIGDERSON, A. 1987. How to play ANY mental game. In *Proceedings of the Nineteenth Annual Conference on Theory of Computing, STOC'87*, ACM, New York, 218– 229.

[37]GONG, L. 1993. Increasing availability and security of an authentication service. *IEEE J. Select. Areas Commun. 11*, 5 (June), 657–662.

[38]GONG, L. AND SYVERSON, P. 1998.

Fail-stop protocols: An approach to designing secure protocols. In *Dependable Computing for Critical Applications 5*, R. K. Iyer, M. Morganti, W. K. Fuchs, and V. Gligor, Eds., IEEE Computer Society Press, New York, 79–99.

[39]HARN, L. 1994. Group oriented (*t*, *n*) digital signature scheme. *IEE Proc. Comput. Digit. Techn. 141*, 5 (September), 307–313.

[40]HERZBERG, A., JAKOBSSON, M., JARECKI, S., KRAWCZYK, H., AND YUNG, M. 1997. Proactive public-key and signature schemes. In *Proceedings of the Fourth Annual Conference on Computer Commu-nications Security*, ACM SIGSAC, ACM, Zurich,¨ 100–110.

[41]HERZBERG, A., JARECKI, S., KRAWCZYK, H., AND YUNG, M. 1995. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology—Crypto'95, the Fifteenth Annual International Cryptology Conference, Proceedings*, *Lecture Notes in Computer Science*, vol. 963, D. Coppersmith, Ed., Springer-Verlag, Berlin, 457–469.

[42]IYENGAR, A., CAHN, R., JUTLA, C., AND GARAY, J. 1998. Design and implementation of a secure dis-tributed data repository. In *Proceedings of the Fourteenth IFIP International Information Security Conference (SEC'98)*, International Federation for Information Processing, TC11: Security and Protection in Information Processing Systems, Elsevier Science, Vienna, 123–135.

[43]JARECKI, S. 1995. Proactive secret sharing and public key cryptosystems. M.S. Thesis, Depart-ment of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass.

[44]JUELS, A. AND BRAINARD, J. 1999. Client puzzles: A cryptographic countermeasure against con-nection depletion attacks. In *Proceedings of the 1999 Network and Distributed System Security Symposium*, Internet Society, San Diego, 151–165.

[45]KARN, P. AND SIMPSON, W. 1997. The Photuris session key management protocol. Internet Draft: draft-simpson-photuris-17.txt.

[46]KAUFMAN, C. 1993. DASS: Distributed authentication security service. Request for Comments 1507.

[47]KENT, S. T., ELLIS, D., HELINEK, P., SIROIS, K., AND YUAN, N. 1996. Internet infrastructure security countermeasures. Tech. Rep. 8173, BBN, January.

[48]KIHLSTROM, K. P., MOSER, L. E., AND MELLIAR-SMITH, P. M. 1997. Solving consensus in a Byzantine environment using an unreliable fault detector. In *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS'97)*, Hermes, Chantilly, France, 61–76.

[49]KOCHER, P. C. 1998. On certificate revocation and validation. In *Financial Cryptography, the Second International Conference (FC'98), Proceedings*, *Lecture Notes in Computer Science*, vol. 1465., R. Hirschfeld, Ed., Springer-Verlag, Berlin, 172–177.

[50]KORNFELDER, L. M. 1978. Toward a practical public-key cryptosystem. Bachelor's Thesis, Depart-ment of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass.

[51]LABOVITZ, C., MALAN, G. R., AND JAHANIAN, F. 1997. Internet routing instability. In *Proceedings of the Annual Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'97)*, ACM, Cannes, 115–126.

[52]LAMPORT, L. 1978. Time, clocks and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (July), 558–565.

[53]LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. 1992. Authentication

in distributed systems: Theory and practice. *ACM Trans. Comput. Syst. 10*, 4, 265–310.
MALKHI, D. AND REITER, M. 1998a.
Byzantine quorum systems. *Distrib. Comput. 11*, 4, 203–213. MALKHI, D. AND REITER, M. 1998b. Secure and scalable replication in Phalanx. In *Proceedings of the Seventeenth Symposium on Reliable Distributed Systems*, IEEE Computer Society, West Lafayette, Ind., 51–58.

[54]MAURER, U. 1996. Modeling a public-key infrastructure. In *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS'96)*, *Lecture Notes in Computer Science*, vol. 1146, E. Bertino, H. Kurth, G. Martella, and E. Montolivo, Eds., Springer-Verlag, Berlin, 325–350.

[55]MCDANIEL, P. AND RUBIN, A. 2000. A response to "Can we eliminate revocation lists?." In *Finan-cial Cryptography, the Fourth International Conference (FC'00), Proceedings*, *Lecture Notes in Computer Science*, vol. 1962, Y. Frankel, Ed., Springer-Verlag, Berlin, 245–258.

[56]MEADOWS, C. 1999. A formal framework and evaluation method for network denial of service. In *Proceedings of the Twelfth IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, Mordano, Italy, 4–13.

[57]MEADOWS, C. 2001. A cost-based framework for analysis of denial of service in networks. *J. Com-put. Sec. 9*, 1/2, 143–164.

[58]MILLEN, J. K. 1992. A resource allocation model for denial of service. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland, Calif., 137– 147.

[59]MILLEN, J. K. 1995. Denial of service: A perspective. In *Dependable Computing for Critical Ap-plications 4*, F. Cristian, G. L. Lann, and T. Lunt, Eds., Springer-Verlag, Berlin, 93–108.

[60]MYERS, M. 1998. Revocation: Options and challenges. In *Financial Cryptography, the Second International Conference (FC'98), Proceedings*, *Lecture Notes in Computer Science*, vol. 1465, R. Hirschfeld, Ed., Springer-Verlag, Berlin, 165–171.

[61]MYERS, M., ANKNEY, R., MALPANI, A., GALPERIN, S., AND ADAMS, C. 1999. X.509 Internet public key infrastructure online certificate status protocol (OCSP). Request For Comments 2560.

[62]PENSSL PROJECT. Available at
http://www.openssl.org.

[63]PPLIGER, R. 1999. Protecting key exchange and management protocols against resource clogging attacks. In *Proceedings of the IFIP TC6 and TC11 Joint Working Conference on Communications*