ARECA: A Highly Attack Resilient Certification Authority

Jiwu Jing
The State Key Lab of Information Security, Graduate School of CAS
19A Yuquan Road
Beijing 100039, China
Jing@ieee.org

Peng Liu
School of Information Science and Technology, The Pennsylvania State University
University Park, PA 16802
pliu@ist.psu.edu

Dengguo Feng
The State Key Lab of Information Security, Institute Of Software, CAS
4 South Fourth Street, Zhongguancun
Beijing 100084, China
feng@is.iscas.ac.cn

Ji Xiang   Neng Gao   Jingqiang Lin
The State Key Lab of Information Security, Graduate School of the Chinese Academy of Sciences
19A Yuquan Road, Beijing 100039, China

ABSTRACT

Certification Authorities (CA) are a critical component of a PKI. All the certificates issued by a CA will become invalid when the (signing) private key of the CA is compromised. Hence it is a very important issue to protect the private key of an online CA. ARECA systems, built on top of threshold cryptography, ensure the security of a CA through a series of defense-in-depth protections. ARECA systems won't be compromised when a few system components are compromised or some system administrators betray. The private key of a CA is protected by distributing different shares of the key to different (signing) components and by ensuring that any component of the CA is unable to reconstruct the private key. In addition, the multi-layer system architecture of ARECA makes it very difficult to attack from outside. Several threshold-cryptography-based methods are proposed in the literature to construct an intrusion tolerant CA, and the uniqueness of ARECA is that it engineers a novel two phase signature composition scheme and a multi-layer CA protection architecture. As a result, ARECA is (a) practical, (b) highly resilient to both insider and outsider attacks that compromise one or more components, and (c) can prevent a variety of outside attacks.

General Terms

Algorithms, Security.

Keywords

Attack Resilience, Intrusion tolerance, CA, Digital Signature, RSA.

1. INTRODUCTION

A PKI (Public Key Infrastructure) is based on algorithms of public key cryptography. In a PKI system, a CA is the center of trust (within a domain), and most, if not all, of the security properties of the communications among computers are dependent on the *digital certificates* issued by a CA. A digital certificate issued by a CA is the binding of an identity and a specific public key which is signed by the CA's private key. Generally speaking, there are two steps for one party A to verify another party B's identity. First, verifying whether the signature on B's certificate is correct. The signature can only be produced by a CA because only the CA knows the signing key. Second, verifying whether party B knows the private key corresponding to the public key listed in the certificate. If the two-step verification succeeds, then the identity data contained in the certificate should be viewed as the true identity of party B. Therefore, the private key of a CA is the core of CA security, and protecting the private key from being compromised is the foundation of the security of the whole CA domain.

As a key piece of an infrastructure providing security services, the security of CA has been paid substantial attention. If a CA itself is not secure, the applications depending on the CA will not be secure either. For example, for an application that builds its security on top of the trustworthiness of certificates, if the CA that issues the certificates is compromised, then the security of the application is primarily lost since certificates can be faked by the attacker easily and no certificate may be trusted any longer. Hence, CA security is the key of PKI security and CA systems should be well protected.

However, protecting a CA is not an easy job due to several reasons. First, CA is a major target of hackers, since the corresponding payoff can be very significant. Organized hackers who attack the network for national and group interests can be very well equipped. As PKI is more widely deployed, PKI becomes more important a target to the hackers. Since the security of PKI is dependent on the security of the CAs, the attacker typically needs to attack and break into a CA first in order to conquer the key part of an application.

Second, existing networks and computers are still very vulnerable to attacks. No operating system or network protocol can guarantee that they are 100% secure. New bugs are identified almost every day which can allow the attacker to break into a computer system before it is patched accordingly. Hence, the PKI has to secure itself instead of relying on the security of the underneath network protocols and operating systems.

Third, CA systems are designed to stay online and provide continuous services even under sustained attacks. This makes securing CA even more challenging. After an intrusion is detected and part of a CA is infected, shutting down the whole CA for an offline repair may not be affordable. For example, a certificate may need to be revoked during the repair and it should immediately be revoked. If the CA cannot revoke the certificate right away, it is possible that some dangerous activities could be performed using the damaged certificate, which can jeopardize the security and trustworthiness of the whole PKI. Hence, when an intrusion occurs, recovering the signing key may need to be done as soon as possible in order to prohibit criminals from performing illegal operations. If the key cannot be recovered in time, big loss could be caused. Therefore, we must ensure not only the confidentiality and integrity of the private key of a CA, but also the availability of the CA service.

Fourth, existing CA systems are very vulnerable to malicious insider threats. Experiences with insider crimes show that many CAs can be attacked by malicious insiders, e.g., the administrators and operators involved in a certificate issuing or revoking process. Although many insider threats can be countered by a strict PKI management policy/system, the insiders typically have a better chance to break into a CA since they could exploit the operational mistakes and management negligence within a CA which outsiders usually are unable to take advantage of. Moreover, because of the importance of a CA, the adversary may want to invest a lot of money to bribe an insider who may not be able to resist the corresponding temptation.

In summary, although no CA can prevent all the possible attacks, a variety of important web/network applications, such as e-commerce and e-government, are based on the assumption that certificates are trustworthy and these applications require CA systems be able to deliver correct services even under partially successful attacks. In other words, attack resilient CA systems are of urgent need. If a hacker can break into a CA and get its private key, users must be seized with panic. If a large-scale CA is conquered by the attacker, a big society can be frightened and paralyzed. If a hacker can make a CA stop delivering services by infecting part of the CA (without stealing its private key), the digital society may still suffer a lot of security loss or denial-of-service.

The goal of this paper is to develop a highly attack resilient CA, called **ARECA**. ARECA assumes a centralized CA. ARECA is designed to counter two types of attacks: (a) an outsider attack may break into a CA; the hacker could obtain the server resources and thus find the private key of CA or find the way to lead to or make use of the private key; (b) An employee, who has control of part of a CA, wants to figure out what is the private key of the CA. Handling denial-of-service attacks caused by too many legitimate or spoofed CA service requests is out of the scope of this paper.

The basic idea of ARECA is to distribute the certificate signing task from a single CA signing server to a set of *share servers* and a set of *combiners* in such a way that (a) each share server knows a share of the private key but not the whole key; (b) when no combiner is hacked, (b1) compromising any/all share servers will not disclose the private key (when a server is compromised, we assume the attacker knows every thing the server knows); (b2) the CA will not stop running when more than t out of k share servers are running correctly; (d) even when some combiners and some share servers are both compromised, the attacker in general still has a small chance to get the private key.

ARECA shows that a RSA-based CA can be made highly attack resilient. Although ARECA can work off-line in a safe environment to obtain more security, ARECA is designed to deliver non-stop, online CA services. The whole ARECA system is composed of several servers, but none of them can cause the system to disclose its private key. The design goal of ARECA is:

Security. It is impossible for the attacker to get the private key when no combiner is hacked no matter how many share servers are hacked. It is very difficult for the attacker to get the private key even when some combiners and some share servers are issuing a conspiracy attack.

Resilience. Non-stop, correct certificate signing services can be continuously delivered even when some share servers or combiners are hacked.

Simplicity. (a) The certificate signing protocol is simple and efficient. (b) All the share servers are doing the same kind of computation, so do all the combiners. The computation complexity of share servers and combiners are similar.

Flexibility. The CA can be flexibly configured in terms of who hold which key shares, and who do which tasks.

Accountability. The audit system is able to trace attacks and/or failures.

Ease of Management. The clients need to know nothing about the key share distribution scheme. A share server needs to know nothing about "I should collaborate with which share servers?" Adding a new share server or a new combiner are both easy.

Low cost. The system can be built on top of COTS (commercial off the shelf) components.

Several threshold-cryptography-based methods are proposed in the literature to construct an intrusion tolerant CA, and the uniqueness of ARECA is that it engineers a novel two phase signature composition scheme and a multi-layer CA protection architecture. As a result, ARECA is (a) practical, (b) highly resilient to both insider and outsider attacks that compromise one or more components, and (c) can prevent a variety of outside attacks.

The rest of the paper is organized as follows. In Section 2, we discuss a set of existing threshold cryptography methods. In Section 3, we present the design principle of ARECA. In Section 4, we present the system architecture of ARECA. We present the implementation of ARECA in Section 5. In Section 6, we address the performance of ARECA. In Section 7, we discuss some related work. We conclude the paper in Section 8.

2. EXISTING THRESHOLD METHODS

The key innovation of ARECA is motivated by the limitations of existing threshold cryptography methods in engineering intrusion tolerant CAs with respect to not only security, but also scalability, efficiency, simplicity, flexibility and cost.

Existing private key protection, for the purpose of intrusion tolerance, utilizes threshold cryptography techniques. (A more comprehensive introduction of threshold cryptography has been given in [4].) This paper focuses on the threshold cryptography schemes based on RSA. The computation of RSA requires three parameters, N, d, e, where N is the modulus; e is an open exponent used to encrypt messages or verify signatures; and d is a private exponent used to decrypt or sign messages. N and e can be open to the public. d is the *private key* that needs to be protected. The easiest way is to protect d in an intrusion tolerant way is to split d into the sum of t random numbers. Wu et al. [2] present a generic intrusion tolerance scheme based on this idea. The scheme is simple and can be proved secure. It divides private key d into the sum of several numbers $d=d_1+d_2+\ldots+d_t$. And then it distributes d_i (called a *share* of d) to the ith *signing server* (server for short). When a signature is needed, the client sends the HASH (or digest) of the message that needs to be signed, denoted M, to these t servers. Then each server sends the computation result $M_i = M^{d_i}$ *(Mod N)* back to the client, where N is the modulus part of the RSA public key (N,e). The client then computes

$$S = \prod_{i=1}^{t} M^{d_i} = M^{\sum_{i=1}^{t} d_i} = M^{d} \ (Mod\ N)$$

to get the desired result. Note that when one server betrays, it only knows d_i, and it cannot compute other d_j based on d_i. Meanwhile, a betrayal could eavesdrop $M_j = M^{d_j}$ *(Mod N)* but the complexity of computing d_j from M_j is similar to the complexity to RSA (i.e., the complexity of the *discrete logarithm* problem). Because d is not in any of the t signing servers, it is impossible to directly obtain d. Therefore, it is impossible for a betrayal to generate a faked signature.

In order to apply this scheme in the real world, [2] generates duplicate configuration by dividing d in multiple independent ways, that is, by randomly generating several *key share groups* of numbers where the sum of each group is exactly d. For example,

The 1st group: $d=d_{11}+d_{12}+\ldots+d_{1t}$

The 2nd group: $d=d_{21}+d_{22}+\ldots+d_{2t}$

…

Then [2] distributes the shares of d to different servers, and each server will get multiple shares, but any two shares on a server never come from the same group. For example, in a situation where there are 4 servers and $t=3$, the distribution plan can be as follows:

Table 1. Redundant Key Share Distribution

Server 1	Server 2	Server 3	Server 4
d_{11}	d_{12}	d_{13}	d_{13}
d_{21}	d_{21}	d_{22}	d_{23}

When the client needs M to be signed, it designates t undamaged servers. It then notifies these servers which group of shares should be used. The servers then compute the partial signatures using the corresponding shares. Moreover, [2] describes such techniques as how to use a partial signature scheme to detect the servers that do not work correctly, either due to attacks or failures. It is able to verify the correctness of the server based on a specific crypto scheme developed by Frankel et al. in [1].

The benefits of the above scheme are obvious. It has a simple structure and provides exceptional security. The use of redundant secret sharing makes the scheme easy to maintain and more fault/intrusion tolerant. However, [2] has the following limitations.

First, as the number of servers increase, the number of key shares for a server to manage will increase in a non-linear speed. For example, by a rough calculation, if the number of the servers is k, then the number of the private keys stored in each server will be increased to C_{k-1}^{t-1} in the worse case. The reason is as follows:

- We assume the following **intrusion tolerance property** is required to hold all the time: an ARECA system will still function correctly when no more than k-t servers stop to work correctly due to attacks or failures. That is, any *t server group* must be able to correctly sign a message.
- To not violate the above property, every t server group needs to completely store at least one key share group, so in total C_k^t key share groups are needed, although two t server groups may have the same key share group.
- Since the resilience will decrease when two t server groups have the same key share group, we consider the more resilient case where (a) any two t server groups do not use the same key share group; and (b) any two key share groups do not share any key shares. As a result, on average $tC_k^t / k = C_{k-1}^{t-1}$ key shares need be stored on each server.

In general, when the values of k and t are comparable, the number of key shares stored in each server will increase in a nonlinear fashion when a new server is added. When $k=5$, $t= 3$, using such a calculation, the number of key shares stored in each server shall be 6. When $k=6$, $t= 3$, using such a calculation, the number of key shares stored in each server shall be 10. When $k=7$, $t= 4$, the number of key shares stored in each server shall be 20. Note that [2] does not provide the private key distribution algorithm when k is much bigger.

Second, in [2], a client needs to ask exactly t servers in order to sign a message. The server selection procedure is inefficient and vulnerable to attacks and failures. To get a message signed, the client needs to first select t servers before any partial signature is computed. Then the client needs to find the key share groups that match these t servers. For example, in the example shown in Table 1, key share group (d_{11}, d_{12}, d_{13}) matches (Server 1, Server 2, Server 3) but not (Server 2, Server 3, Server 4). For this purpose, every client needs to know which share server has which kind of key shares. When one of the servers fails to compute the correct partial signature (due to attacks or failures), the whole signature generation process has to re-start from the server selection step. When a new server is added, the knowledge of each client needs to be refreshed, since the clients need to know the group membership of the new server as well.

It is not ideal in an intrusion/fault tolerant system to have the client choose the signing servers. We want our scheme to have a feature similar to Shamir's secret sharing scheme [6] where the private key can be generated by any *t* key shares. However, to achieve this, the private key has to be first reconstructed in Shamir's secret sharing scheme, which is not what we want because this can make the CA much less attack resilient. We do not want the private key *d* to be reconstructed at any point of time under any circumstances.

Several papers have discussed how to design an attack resilient CA based on Shamir's secret sharing scheme [6]. For example, such a design is described in [1] and [3]. In particular, given a polynomial $f(x)=\sum_{i=0}^{t-1} a_i x^i$, using the *interpolation theorem* of LaGrange, we get:

$$f(x)=\sum_{i=1}^{t}(f(x_i)\prod_{j=1,t}^{j\neq i}\frac{x-x_j}{x_i-x_j}) \qquad (1)$$

By randomly selecting *t* pairs of $\{x_i, f(x_i)\}$, we have:

$$a_0=f(0)=\sum_{i=1}^{t}(f(x_i)\prod_{j=1,t}^{j\neq i}\frac{-x_j}{x_i-x_j}) \qquad (2)$$

We can let a_0 be the private key *d*. As a result, signing a message (digest) *M* can be done as follows.

$$M^d=M^{f(0)}=\prod_{i=1}^{t}M^{f(x_i)\prod_{j=1,t}^{j\neq i}\frac{-x_j}{x_i-x_j}}=\prod_{i=1}^{t}M^{b_i} \qquad (3)$$

Here, $b_i=f(x_i)*c_i=f(x_i)\prod_{j=1,t}^{j\neq i}\frac{-x_j}{x_i-x_j}$ (4)

Thus, we can split and distribute *d* to *k* servers ($k\geq t$). When *M* needs to be signed, we let each server compute M^{b_i} and then let a combiner multiply these results to get the value of M^d without reconstructing the private key *d* during any step of this process.

Because there are division operations when calculating b_i, b_i may not an integer and may be a number in the form of *p/q* where *p* and *q* are two integers. As a result, the process of computing M^{bi} may include the extraction of a root. For example, $M^{8/3}$ requires extracting the cube root. However, we know it is extremely difficult to compute the root extraction out of a large integer. In order to solve this problem, such algebra structures as *fields* or *rings* are usually needed to help us to find the conditions that can ensure that x_i-x_j will always have a mathematical *inverse* with respect to a specific ring. In particular, [1] identifies the following condition:

- [**The Inverse Condition**]: $M^{(bi \bmod v)}$ will not need any root extraction if (a) *v* is a prime number, or (b) the values of all the x_i are chosen in such a way that the value of the rank-*t* *Vandermonde* matrix composed by all the x_i is relatively prime to *v*.

Even if the Inverse Condition is satisfied, since $M^{(bi \bmod v)}$ will be computed by each share server instead of M^{bi}, the production calculated by the combiner is actually not M^d but the following:

$$\prod_{i=1}^{t}M^{b_i(\bmod v)}=M^{d+wv}$$

Here the value of *w* is unknown but *w* is typically not zero, hence, removing the impact of *v* on the signature is fairly difficult.

As we know, $M^{\Phi(N)}=1 \bmod N$, so many people may want to let $v=\Phi(N)$ in order to solve the above problem. However, it is found that letting $v=\Phi(N)$ will make the selection of x_i greatly restricted by the Inverse Condition. Moreover, if we let $v=\Phi(N)$, the share servers must know how to compute the inverse of a number modulo $\Phi(N)$. When an element *o* and the inverse of it modulo $\Phi(N)$, i.e., o^{-1} , are known, the attacker can compute the value of $\Phi(N)$. However, if $\Phi(N)$ is known, then the private key can be computed. So letting $v=\Phi(N)$ is clearly not a secure scheme.

Frankel et al. [1] proposes to (a) let a_i be a number within set $\{0, L,...,2L^3N^{2+e}t\}$, where $L=k!$, and (b) choose x_i from $[1, 2,..., k-1]$. Since when divided by *L*, every $f(x_i)$ can get an integer result, the computation of b_i does not need the inverse-operation anymore, and the computation can be performed with integers only. This scheme can work with the general RSA algorithm and does not need strong-prime numbers. However, since the selection of parameters is tightly restricted, it is complicated to prove its security. The security of the above scheme is proved in [1]. In [1], when b_i is being computed by a share server, since the computation of b_i is dependent on the selection of x_i, the share server must know who his or her collaborators are. Hence, this scheme has the similar share server selection problem as [2].

[3] uses the scheme of RSA strong-primes, where its secret primes are: $p=2p'+1$, $q=2q'+1$. In [3], all the relevant interpolation computations are processed within the *ring* of modulo $m=p'q'$. Since $M^{4m} \bmod N = 1$, when being computed in a distributed way, one additional square operation is needed, then the combiner will compute the square of the result from each share server, and the result will be $M^{4\Delta(gm+d)}$, where $\triangle=(k!)^2$ and *g* is an integer. In [3], c_i (see equation (4)) is handled by the combiner. Hence, the share server selection problem is avoided. However, the computation complexity of the combiner is substantially increased. In [3], the combiner must compute:

$$W=y_{i_1}^{2\lambda c_{i_1}}y_{i_2}^{2\lambda c_{i_2}}...y_{i_t}^{2\lambda c_{i_t}}$$

Where y_i is the result of each Share Server, and $\lambda=k!$

Finally, both [1] and [3] provide good partial signature verification methods, which we will not discuss in detail here.

3. ARECA DESIGN PRINCIPLE

In this section, we propose an innovative *two phase signature composition* approach to enhance the resilience of a CA and overcome the three limitations of [2] which we identified in Section 2. In particular, we first present the two phase signature composition approach; then we analyze its security and discuss its advantages.

3.1 Two Phase Signature Composition: An Overview

ARECA implements the two phase signature composition scheme as Figure 1 shows. The *RA Agent* is the interface between the *Registration Authority* (RA) and the signature composition subsystem. The RA talks to the applications (and customers) directly and is responsible for collecting, preparing and verifying the materials (or information) contained in each certificate. The RA Agent has secure communications with the RA and it also verifies the RA's signatures (when receiving a *certificate signing request* from the RA). In the mean time, the RA Agent is also an interface to the outside network. Customers may directly apply for a certificate through this interface as well.

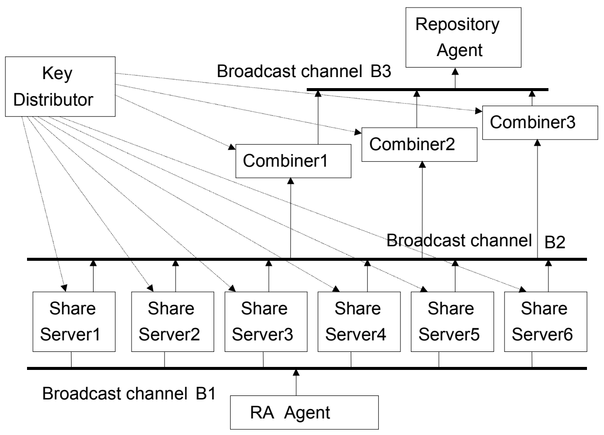

Figure 1. Two phase signature composition

Share servers 1-n are used to compute partial signatures (on a certificate). Each share server (server for short) holds some shares of the CA's private key. Each server has its own ID. The RA Agent is connected to these servers through broadcast channel B1. We assume B1 is of high bandwidth and reliable since it is typically implemented via a high speed LAN.

The *combiners* are responsible for the second phase of signature composition. Multiple combiners are used to increase fault tolerance and resilience. The output of a combiner is the real signature if there are no mistakes.

To achieve more security, an administrator may be assigned to a server or combiner. In this case, the RA needs not to be completely trusted, since the administrator assigned to a server or combiner may double-verity the certificate materials before allowing the server or combiner to sign or compose the certificate.

The *Key Distributor* is responsible for distributing the key shares to both the share servers and the combiners. It is normally off-line. When the CA is initialized or when an old private key is replaced by a new one, the Key Distributor may become on-line. The connection between the Key Distributor and a server or combiner can be the classical *key-transport* method.

The *Repository Agent* is the interface to the *Certificate Database* (not shown in Figure 1 but shown in Figure 2) where the customers can receive their certificates and query other parties' certificates. The Repository Agent is also responsible for failure/attack detection and response. A combiner will notify the Repository Agent at once whenever its verification operations fail. The Repository Agent will then check (the recent computation results of) the servers and the other combiners to locate the problem.

3.2 Two Phase Certificate Signing

We first present a basic certificate signing protocol for clarity. Then we show how this basic signing protocol can be improved.

[The basic signing protocol]

Assumptions:

(a) We denote the private key of the CA as *d*; and the pubic key as *e* and *N*. Share server *i* is denoted as S_i. (b) We assume there are *k* share servers. (c) We assume at least *t* share servers are needed to generate a signature. (d) We assume there can be multiple combiners.

Preparation:

(a) The Key Distributor randomly chooses *k* random numbers $d_1, d_2, ..., d_k$, the value of each of them should be less than *d/t*. These numbers can be far less than *d* but their length should not be less than half of the length of N to protect combiners from knowing more than half bits of d.

(b) The *k* random numbers are exactly the key shares. They are distributed to the *k* share servers as follows: number d_j is sent to server S_j. As a result, each share server will hold only **one** key share.

(c) A server group composed of *t* share servers is called a *t-group*. Out of *k* servers, maximally C_k^t *t*- groups can be formed (note that two *t*-groups can partially overlap). For each *t*-group *j*, the Key Distributor computes a specific *combiner key share*, denoted c_j, as follows:

$$c_j = d - (d_{i_1} + d_{i_2} + d_{i_3} + ... + d_{i_t})$$

Here $i_1, i_2, i_3, ..., i_t$ indicate exactly the set of share servers belonging to the *t*-group. In other words, the value of c_j is determined by the sum of the set of key shares held by *t*-group *j*. We call $(i_1, i_2, i_3, ..., i_t)$ the *index* of c_j. Since there are C_k^t *t*-groups, the Key Distributor will generate C_k^t combiner key shares. For example, when *k=5, t=3*, there are totally 10 *t*-groups, and $c_1, c_2, ..., c_{10}$ will be computed.

(d) The Key Distributor then distributes the set of combiner key shares evenly over the set of combiners. For example, when *k=5, t=3,* if there are 2 combiners, then each combiner will get 10/2=5 combiner key shares. When we distribute a combiner key share c_j to a combiner, we sent the index of c_j to the combiner as well.

The 1st step: When a certificate (denoted *cert*) needs to be signed, a unique task number, denoted *Task(cert)*, will be assigned to this signing task. Every RA Agent has a unique identifier. (Note that there can be multiple RA agents.) Every RA Agent also maintains a *task serial number*. Whenever a RA agent is assigned with a new signing task, the agent's task-serial-number will be increased by 1, and the signing task's task number is the combination of the agent's identifier and the agent's current task-serial-number. This task number is unique in the whole CA.

The 2nd step: Share server S_i computes its busy factor $F_i(Task(cert))$, which is the production of the number of tasks within the task queue of S_i and the *calculate speed* of S_i. The calculation speed of S_i is determined by the time it takes S_i to calculate one standard module exponentiation.

The 3rd step: Each share server broadcasts its busy factor (and the associated task number) to the other servers. For each server, if it is idle, it will jump to the 5th step. Otherwise, it will sort the busy factors of the other servers.

The 4th step: For each server S_i, if there are at least *m* other servers which are not as busy as S_i, S_i will discard the new task from its task queue. Otherwise, S_i will go to the next step when it finishes all the tasks in its queue The value of *m* is determined based on the security and performance requirements of the CA. However, *m* should be no less than *t*.

The 5th step: S_i will compute $y_i = M^{d_i}$, which is $(HASH(cert))^{d_i}$. Here HASH is a hash function. Then S_i will broadcast the following message to the set of combiners via channel B2: $\{cert, Task(cert), i, y_i\}$. Here *i* is the identifier of S_i.

The 6th step: When a set of broadcast messages are received, each combiner will do the following: (6.1) it will prepare the set of combiner-key-share indexes, where each index is composed of *t* share server identifiers (e.g., $\{i_1, i_2, i_3, \ldots, i_t\}$). The set of indexes are not difficult to prepare based on the server identifier component of each broadcast message. (6.2) The combiner will then use the set of indexes to match its combiner key shares. If no match is found, it will wait for more results from the set of share servers. (6.3) Otherwise, as soon as a (or another) combiner key share c_j is matched, it will compute

$$R = (HASH(cert))^{c_j} * \prod_{i=i_1}^{i_t} y_i$$

R is indeed the signature needed on the certificate. (6.4) The combiner will use the CA's public key to verify *R*, if the verification succeeds, it will jump to the 7th step; otherwise, the combiner will go back to step 6.3 to find another match until all the matches are tried.

The 7th step: Case the combiner computes a correct *R*, it will send to Repository Agent (a) the certificate, (b) the set of partial signatures involved, (c) the set of share servers involved, and (d) *R* through channel B3.

Case any verification fails, the combiner will report the failure to the Repository Agent. The report message includes (a) the combiner's identifier, (b) the set of partial signatures involved, and (c) the set of share servers involved. Each failure report will be analyzed by the Repository Agent as soon as possible to detect the corresponding faults or attacks.

The 8th step: When receiving a signature *R*, the Repository Agent will verify *R* again before sending *R* and all the relevant information to the customer and the Certificate Database for future reference. As soon as *R* is verified, it will notify the other combiners to stop processing this signing task. Whenever a "stop" notice is received by a combiner, the combiner will discard the data related to the task.

3.2.1 Security Analysis

In this section, we focus on the impact of combiners on attack resilience. (The other advantages of the ARECA design will be mentioned in Section 3.2.3.) We will show that the use of combiners in general makes the CA more attack resilient. In particular, we will analyze three cases: (A) some share servers are broken, but no combiner is broken; (B) some combiners are broken, but no share server is broken; (C) some share servers and some combiners are broken at the same time. We assume the attacker will know all the secret kept on a broken machine.

In case A, ARECA is more resilient than the scheme of Wu et al. [2]. The only information on the network is M^{di}. Although a malicious share server can know many of M^{di}, using M^{di} to infer d_i is as difficult as breaking RSA. When the set of broken servers does not contain a key share group of size *t*, [2] is as resilient as ARECA. However, when the set of broken servers contains a key share group, [2] is broken because the attacker now knows the private key *d*. In contrast, in ARECA, the attacker cannot get *d* even if he or she compromises all the share servers. In particular, since d_{ix} is randomly chosen, d_{ix} has no relationship with *d*. Therefore d_{ix} won't expose any information about secret key *d*, that is, the conditional entropy $H(d|d_{ix})=H(d)$. In addition, because d_{ix} and d_{jx} are independent random variables when $i \neq j$, so $H(d|d_{ix},d_{jx}) = H(d|d_{ix}) = H(d|d_{jx}) = H(d)$. In other words, disclosure of multiple d_i won't expose any information about *d*.

In case B, first of all, the combiners are more difficult to be broken by an (outside) attacker than the share servers due to several reasons. (a) Since the combiners only accept messages from the share servers, the attacker has to first break into a share server in order to break into a combiner. (b) Since the combiners and the share servers run different services, and they are suggested to be protected in very different ways (e.g., combiners and share servers can use different operating systems), so the attacker cannot use the same technique he or she used to break into a share server to break into a combiner.

Second, since the output of any combiner is the signature which will not disclose any information about the private key, even if a malicious combiner can eardrop and know all the outputs of the other combiners, the malicious combiner cannot gain any additional information about *d* from these outputs.

Third, in the following, we show that when the attacker breaks into a combiner, he or she cannot calculate the value of *d*. We assume the worst case, that is, the broken combiner holds all the combiner key shares. As a result, the attacker can get C_k^t linear equations and each such equation looks like: $c_j = d - (d_{i_1} + d_{i_2} + d_{i_3} + \ldots + d_{i_t})$, where c_j is known but none of the other variables is known to the attacker. Here the resilience is determined by whether the attacker can use the C_k^t combiner key shares to compute *(k+1)* variables: the *k* share server key shares and *d*. For example when *k=5* and *t=3*, there are 10 linear equations and 6 variables. The coefficient matrix (of the 10 linear equations) is as follows:

$$\left[\begin{array}{ccccc} 1 & -1 & -1 & \ldots & 0 \\ 1 & -1 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & & \\ 1 & 0 & 0 & \ldots & -1 \end{array}\right]$$

The matrix has *k+1* columns and C_k^t rows, where all the elements in the first column (i.e., the column for variable *d*) are 1, and every row has exactly *t* non-zero elements whose values are all *-1*. It can be seen that the *rank* of this coefficient matrix is in fact the matrix's row-rank or column-rank. We can see that the first column is the linear combination of the other *k* columns since the first column is the sum of the other *k* columns divided by *–t*. According to this observation, the rank of this matrix is less than or equal to *k*. Since *k+1* variables cannot be computed by a group of linear equations whose rank is less than or equal to *k*, hence, we reach the conclusion that the attacker will not be able to get *d*.

Fourth, based on the above discussion, it is clear that even if all the combiners conspire to attack the system, they cannot get the value of *d*.

In case C, we found that some malicious combiners and some malicious share servers may be able to work together to get the private key *d* due to the following observation:

Observation 1. Although a malicious combiner can know neither d_i nor d_j based on M^{di} or M^{dj}, the malicious combiner may be able to computer the value of $d_i - d_j$. To illustrate, consider the following two equations:

$c_1 = d - (d_1+d_2+d_3)$

$c_2 = d - (d_2+d_3+d_4)$

It is clear that $d_4 - d_1 = c_1 - c_2$. So if the attacker knows $c_1 - c_2$, then the attacker will know $d_4 - d_1$.

Based on the above observation, a combiner who knows $d_i - d_j$ and the share server who holds d_i can conspire to get the value of d_j. We call this attack the **conspiracy attack**.

It is clear that we cannot prevent all conspiracy attacks because when all the combiners and all the share servers are controlled by the attacker, the attacker knows *d*. However, we can make ARECA very resilient to the conspiracy attack by properly distributing the combiner key shares. In particular, we found that a set of specific resilience conditions which can ensure a high level of resilience can be identified and may be satisfied by some combiner key share distribution schemes. For example, when there is only one combiner broken, a resilience condition is as follows.

1-Combiner Resilience Condition. When conspiring with up-to *t-1* share servers, a combiner cannot get the value of *d* if there are variables from more than *t* share servers in any equation generated by the linear combination of any group of linear equations owned by the combiner.

Theoretically, this condition will be satisfied when each combiner holds only one combiner key share, which is not difficult to implement. However, the drawback is that too many combiners may be needed to satisfy this condition. We have tried to find better combiner-key-share distribution schemes that satisfy the above condition, but the results are not encouraging. For example, when *k=5* and *t=3*, a simple computer aided distribution scheme shows that 8 combiners are needed to hold all the C_5^3 key shares and to satisfy the resilience condition. A careful manual distribution may lower the number, but still 7 combiners will be needed in this example. Fortunately, we found that the number of combiners (that are needed to satisfy the resilience condition) can be dramatically reduced when we store multiple key shares on each share server. We will present the details of this idea in the next section.

Finally, it should be noticed that when multiple combiners conspire with multiple share servers, the corresponding *n*-combiner resilience conditions can be stricter. Although *n*-combiner resilience conditions may be defined in a way similar to the 1-combiner resilience condition, how to guarantee that these conditions will be satisfied is beyond the scope of this paper and it part of our future work.

In summary, whether a set of resilience conditions will be satisfied in an ARECA system is highly dependent on how the system distributes its share server key shares as well as combiner key shares. As a result, one key share distribution scheme can be much attack resilient than another.

3.2.2 Supporting Multiple Key Shares on Each Server

In the basic certificate signing protocol, there are totally *k* share server key shares, and each share server has only one key share. In this section, we extend the basic protocol to store multiple key shares on each share server. The advantage of this new protocol is that the number of combiners needed to satisfy the resilience conditions will be much smaller than that of the basic protocol without losing any availability or security. In particular, the new protocol is associated with three design requirements:

- **Availability requirement**. The basic protocol ensures that every group of *t* share servers can find a combiner helping them to generate the signature. We want the new protocol to have the same property.
- **Security requirement**. We want the new protocol to be as secure as the basic protocol, and we still want to ensure that at least *t* share servers are needed to sign a certificate. Therefore, **no** share server can store two or more key shares that will appear in a single linear equation (held by a combiner) that involves a single combiner key share c_j.
- **Resilience requirement**. Every resilience condition that can be satisfied by an implementation (i.e., a setting or a configuration) of the basic signing protocol can be satisfied by an implementation of the new protocol.

In the following, we first present a simple technique to build the new protocol, and show that the simple protocol satisfies all the three requirements. Then we present the protocol used in the ARECA prototype. Finally, we discuss how the simple protocol can be improved.

[The Naive Approach:] Consider the case when k=5 and t=3. In order to satisfy the availability requirement, C_5^3=10 groups of share servers need to be supported by the set of combiners. It is not difficult to see that each share server is involved exactly in C_4^2 = 6 groups. Since each t-group of share servers needs to be supported by a combiner key share, and each combiner key share is associated with a linear equation, so each share server is involved in 6 linear equations. Based on Observation 1, we know conspiracy attacks can succeed because one share server key share may appear in two or more equations. It is clear that every share-server key share appears in only one equation, conspiracy attacks cannot succeed. Therefore, a naïve approach to make sure that every share-server key share appears in only one equation is to let each share server use 6 different key shares in the 6 linear equations the share server is involved in. As a result, each share server will hold 6 key shares.

The naïve approach satisfies the security requirement because any two key shares held by a share server are never involved in the same equation. The naïve approach satisfies the 1-combiner resilience condition since (1) each (original) linear equation has *t+1* variables and (2) the combination of any two equation will result in an equation of *2t* variables. The naïve approach satisfies the resilience requirement since we no longer need 7 combiners to satisfy the 1-combiner resilience condition. In contrast, even having a single combiner holding all the combiner key shares will not violate the resilience condition. (In this case, the combiner will have 10 equations but 31 variables.) Of course, having two or three combiners will also be secure. In summary, after each share server holds more key shares, the total number of the equation keeps the same while the number of variables for the attacker to solve increases dramatically. Hence, although the number of combiners is decreased, the resilience could still be enhanced. For example, even if the single combiner and 2 share servers are broken, the attacker still faces 10 equations and 19 variables.

The ARECA prototype uses 3 combiners and 5 share servers. The 3rd combiner is used for fault/attack tolerance. Each share server stores two key shares. Share server *i* stores d_{i1} and d_{i2}. The linear equations stored in the three combiners are shown in the following table, where (a) S1 represents share server 1; (b) in each column, '1' indicates the first key share of the corresponding server and '2' indicates the 2nd key share of the server; note that two '1's within two columns indicate two different key shares; (c) every row includes the share server key shares involved in a linear equation; for simplicity, the corresponding combiner key share c_j and d are not listed.

Table 2. The Key Share Distribution Scheme of the ARECA Prototype

Combiner 1						Combiner 2						Combiner 3				
S1	S2	S3	S4	S5		S1	S2	S3	S4	S5		S1	S2	S3	S4	S5
1	2	1				1	1		2			1		2	1	
	1	2	1				2		1	1		2	1	1		
		1	2	1			1	1		2		1	2			1
1			1	2		1		2		1			1		1	2
2	1			1		2		1	1					1	2	1

Because we want the length of key shares to be shorter than the length of d, we let the length of all d_{ix} be half the length of N. Therefore, adding a key share has very little complexity or performance impact on a share server. In terms of the combiners, the computation of ascending power needs be performed only once. For mathematical problem, d_i can be set to 200 bits. That is safe for crypto crack. But for RSA algorithm, as all knows *e* and *N*, more bits are necessary. It is proved that more than half length bits of N is not necessary for the security of a small public key e[7]. In addition, since each combiner has only 5 equations, storing two key shares in each share server has very little impact on the match searching time. To achieve very efficient matching, our prototype uses one 32bit integer to "index" the share servers and the key shares involved in an equation. The integer has 4 bytes and each byte has 8 bits. The first byte indicates which share servers contribute their first key shares to the equation; the second byte indicates which share servers contribute their second key shares to the equation, and so on. For example, an equation indexed by integer 10100000, 01000000, 00000000, 00000000 means the following: (a) within the first byte, the first bit is '1', so the first key share of S1 is part of the equation; the second bit is '0', so the first key share of S2 is not part of the equation; the third is '1', so the first key share of S3 is part of the equation; (b) the second byte indicates that only the second key share of S2 is involved in the equation. Accordingly, each combiner does the matching as follows:

- For each signing task, a 32 bit *coverage integer* will be used by the combiner and it is initialized by 32 bit '0';
- When the combiner receives a message from a share server, the combiner will refresh the coverage integer by setting the corresponding bits as '1'. For example, if the share server is S3, and S3 computes the partial signature (included in the message) using the 2nd key share of S3, then the 11th bit of the coverage integer will be set as '1';
- Periodically, the combiner will compute the logical AND of the coverage integer and each *index integer* kept on the combiner. Whenever the result is the same as an index integer, a match is found and the combiner key share associated with the matched index should be used to compose the signature.

3.2.3 The Advantages of Two Phase Signature Composition

Besides achieving more resilience, the two phase signature composition scheme has the following advantages:

- In [1] and [2], a client needs to know exactly which *t* servers can help the client to sign a message. The server selection procedure is complicated and vulnerable to attacks and failures. By contrast, in ARECA, the client is *stateless*; the set of share servers can determine who should help the client by themselves in a self-organizing fashion; and each combiner can easily figure out which *t* partial signatures should be used to compose a signature correctly. As a result, when a new share server joins, the clients' knowledge need not be refreshed.
- In [2], for high resilience, when the number of server increases, the number of key shares for a server to manage will typically increase in a non-linear speed. In ARECA, to achieve the same amount of resilience, much less key shares need be managed by a server since (a) even if two key share groups A and B overlap, the attacker still cannot be benefited if the two specific combiner key shares for the two groups are stored on two combiners. When one of the two combiners is broken, breaking group A and breaking both A and B make no different to the

attacker. (b) Much less key shares need be managed by a server when we allow key share groups to overlap.

- In [2], clients are responsible for composing signatures. By contrast, in ARECA, the combiners will take care of signature composition.

Hence, because of the two-layer key-share-holder structure of ARECA, key management in ARECA is in general much simpler than [2].

4. ARECA SYSTEM ORGANIZATION

The major components of ARECA are shown in Figure 2. Besides the components involved in the two-phase signature composition protocol, we add (a) several audit systems, (b) a *Query System* for the applications (or customers) to query the Certificate Database for certificates and CRLs (Certificate Revoke List), and (c) a set of security controls.

The ARECA architecture has two unique security features:

Multi-Level Security Control. We divide the set of ARECA components into four *levels* or *zones* based on the procedure of signature composition and the relation among these components. The *red* zone, or the top level, includes the Certificate Database, the Mandatory Information Flow Controller, and the Kernel Audit System. The *yellow* zone (i.e., the second level) includes the combiners, the share servers, (part of) the Repository Agent, the monitors, and the Internal Audit System. The *green* zone (i.e., the third level) includes the set of RA agents, and the LDAP/WWW interface to the Certificate Database. The purple zone (i.e., the fourth level) includes the set of RAs, the Query System, and the External Audit system. We also call the application zone the *blue* zone, where all the applications built on top of PKI should stay. Putting the applications into the blue zone doesn't necessarily mean that the applications are not secure; it just indicates that the security of applications is the last thing we consider when developing ARECA.

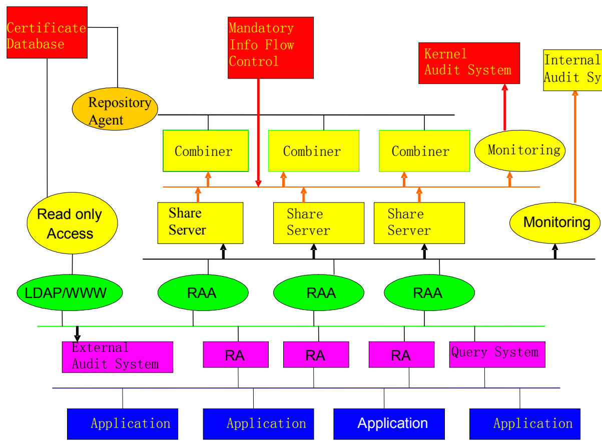

Figure 2. ARCA Components

The multi-level security control requirements of ARECA are as follows:

(a) the connections among zones should be strictly controlled in such a way that the whole CA system can only be compromised zone by zone. That is, a compromised level L component can only be used to compromise a level L+1 component, and it is impossible for the level L component to directly break into any component on a level higher than L+1. For example, a blue zone attacker has to occupy and control at least a component in the green zone in order to break into a component in the yellow zone. (b) Each security level should carry out an independent security strategy. Different zones should be protected in different ways. In this way, the vulnerabilities of one zone are of minimum utility for the attacker to attack another zone.

The benefits of multi-level security control are as follows. (a) Red zone components have the highest security level. These components ensure we will still have the capability to trace (attacks) and search the history activities and results even when all the other lower level components are compromised. (b) Components in the yellow zone must be controlled first in order to attack red zone components (by any means). If no yellow zone components can be controlled by the attacker, the red zone components will be securely "isolated". The yellow zone is the "core" of the CA. As long as majority of the yellow zone components are working correctly, we can guarantee that the whole CA will work correctly and won't disclose any secret. Therefore, we pay most of our attentions to protecting the yellow zone. (c) Even if the attacker compromises some RA components, the attacker cannot compromise a share server without first compromising a RA agent. (d) To control a combiner, a malicious application needs to compromise not only some RA and RA agents, but also some share servers.

ARECA satisfies the multi-level security control requirements by a set of specifically configured firewalls attached directly to each component.

Unidirectional Information Flows: Many information flows among components are unidirectional in ARECA. This is different from many other CA systems. The *unidirectional* design makes ARECA more difficult to break into through a variety of attacks such as worms. When the information flow from component A to B is unidirectional, even if the attacker controls A and can use A to attack B, the attacker cannot know whether his or her attacks are successful. When the attacker does not know the attack result, the attacker in general won't be able to continue his or her attacks. Moreover, when the information flows from one zone to another zone are unidirectional, the attacker even does not know whether the server he or she wants to break into exists. Although unidirectional information flows will cause similar problems to good servers, in ARECA a sender usually needs not to know "who are the receivers?" For example, when a share server sends out a partial signature, no specific receiver needs to be specified, and the set of combiners can figure out who should be the receivers by themselves. Finally, ARECA enforces unidirectional information flows through certain number of specific firewalls attached to some specific components.

5. PROTOTYPE IMPLEMENTATION

The ARECA prototype system is composed of 15 PCs. All CPUs are Intel PPIII 800MHz. The memory of each PC is common. The network interface is 100M 100Base T. Some PCs have two network interface cards (NIC). For simplicity, every PC uses Windows NT Version V4.0, SP6 as the OS. We let k=5 and t=3. We let each share server store two key shares. The key shares are distributed according Table 2.

The prototype is shown in Figure 3. To reduce the cost, we physically divide the whole system into only two networks. However, we logically enforce zone isolation and multi-level security. (To enable independent auditing, we use an Ethernet to connect the set of signing components.) The lines in the figure represent physical links; the computers connected with two links have two independent NIC cards and two IP addresses. The Web RA directly connects to the Internet, through which the ARECA prototype CA has actually been providing free Internet email certificate services for several months to the users in China. Any user can send a certificate signing request to the prototype CA through web site http://pki.is.ac.cn. After a certificate is signed by ARECA, the certificate will be directly sent to the mailbox of the user.

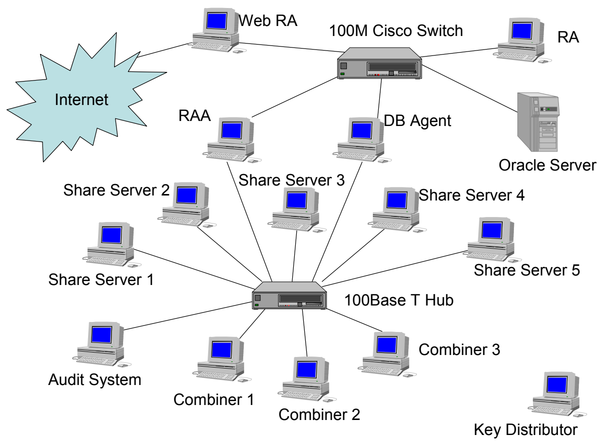

Figure 3. ARECA Installation

In the prototype, we use the UDP protocol for all broadcast communications, and TCP for all point to point communications. All the communications are implemented with WinSock. All the code is written in Visual Studio 6.0. Multi-thread programming is used in all the places where multi-tasks are needed. The whole prototype (including the RAs) consists of about 57,400 lines of source codes contained in 296 separated files of C++.

6. PERFORMANCE

We want to see if the performance of ARECA is comparable to a traditional CA who does not split the private key. ARECA and a traditional CA are different in several ways. (1) As Table 3 indicates, a traditional CA can exploit the Chinese Remainder Theorem (CRT) to make its signing operations two to three times quicker than not exploiting the CRT. However, ARECA cannot exploit the CRT since ARECA does not hold the private key at any time on any place. (2) ARECA needs to compose partial signatures, while a traditional CA does not.

We have done a large number of tests on top of the prototype to evaluate the performance of ARECA. In these tests, the length of the private key *d* is 2048 bits, and the length of a key share is 1024 bits, which is half the length of *d*. We evaluated four ARECA configurations: (a) 1+3, that is, one combiner plus three share servers; (b) 1+5; (c) 3+3; and (d) 3+5. Moreover, for simplicity, we use a random task scheduler instead of busy factors to determine which share servers should generate which partial signatures. In particular, we applied a random 3/4 scheduling on share servers, where each new task arriving at a share server has a probability of 0.75 to be processed by the server and a probability of 0.25 to be discarded by the server, and no task scheduling on combiners. (Note that random task scheduling is more attack resilient than busy-factor-based scheduling since malicious share servers can on longer cheat and 'starve' legitimate servers.)

The performance measurements are summarized in Figure 4, where the Y axle indicates the average time to sign one certificate, and the X axle indicates "how many certificates are being simultaneously signed during that test?" In general, Figure 4 shows that although ARECA is substantially slower than a traditional CA that exploits CRT, when the traditional CA does not exploit CRT, the performance of ARECA is similar to that of the traditional CA; and when there are many share servers, ARECA can even be quicker than the traditional CA. A more detailed comparison between ARECA and a traditional CA is shown in Table 3, where we also address the case when the length of *d* is 4096 bits. When only one certificate is being signed, ARECA is one time slower than a traditional CA without CRT. However, when 20 certificates are being simultaneously signed by a 3+5 ARECA system, ARECA is as quick as the traditional CA.

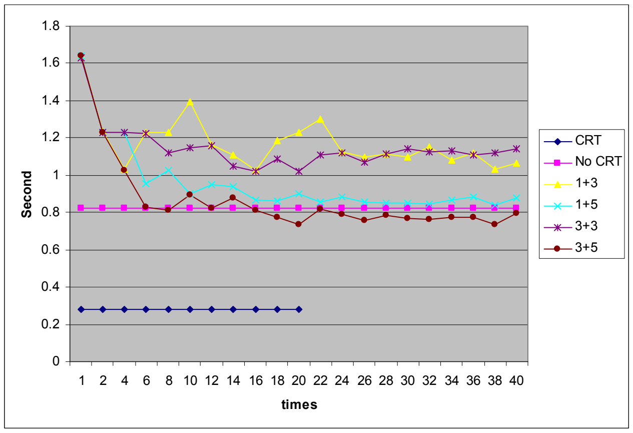

Figure 4. ARECA Performance

We believe after (a) the code of ARECA is tuned and optimized and (b) the task scheduler of ARECA is optimized, ARECA could be significantly quicker than a traditional CA without CRT. The reasons are as follows: (1) ARECA does **parallel** certificate signing, that is, when multiple certificates are being signed, two certificates can be signed by two sets of share servers and two sets of combiners in parallel. Note that this is naturally achieved by (a) the task scheduling algorithm of ARECA and (b) the matching algorithm of combiners. (2) The performance of ARECA may be significantly improved via optimizations of the prototype.

Table 3. Average Signing Time Comparison (in Seconds)

Length of d		2048 bits	4096 bits
Single Machine with CRT	Single cert	0.280	1.652
	20 certs	0.2761	1.6519
Single Machine without CRT	Single cert	0.821	5.888
	20 certs	0.8227	5.8840
ARECA with 5 share servers and 3 combiners	Single cert	1.6420	11.867
	20 certs	0.73805	5.95155
ARECA with 3 share servers and 1 combiner	Single cert	1.64300	11.8670
	20 certs	1.22575	8.28995

Finally, the test results also show that the number of combiners does not have a big impact on the performance of ARECA. For example, Figure 4 shows that 1+3 and 3+3 have no big difference. Two reasons lie under this observation. (1) The combiners do not have a task scheduler, so it is of higher probability that several combiners are signing the same certificate when multiple certificates are being signed. (2) Every signature needs at least 3 share servers, so when there are not many share servers the concurrency degree among combiners is low. Hence, when the number of share servers is significantly larger than that of combiner, the number of combiners should have a bigger impact.

7. RELATED WORK

Besides the threshold cryptography schemes we discussed in Section 2, ARECA is related to COCA [5] since both ARECA and COCA are intrusion tolerant CA solutions. However, although threshold cryptography is used by both ARECA and COCA, they are very different. (a) COCA is a distributed CA where key shares can be located across the Internet, while ARECA is a centralized CA. (b) COCA assumes asynchrony, while ARECA assumes reliable broadcasting on LANs. (c) COCA replicates certificates and focuses on integrating threshold cryptography and Byzantine quorum systems, while ARECA uses a centralized certificate database and focuses on two phase signature composition. (d) COCA is designed to directly integrate any existing threshold cryptography scheme, while ARECA presents a novel approach to engineer existing threshold cryptography schemes in a more resilient and efficient way. (e) Multi-level security control is enforced in ARECA but not in COCA where servers are peers to each other. (f) COCA presents a novel protocol for proactive secret sharing, while ARECA does key share refreshing in an ad hoc way. (g) COCA considers both certificate "query" and "update", while ARECA focuses on certificate "update" only. (h) DDoS defense is an important part of COCA, but is not investigated in ARECA.

In [8], an eight page abstract, we introduced the idea of two-phase signature composition and proposed the basic certificate signing protocol, however, (a) the scheme of storing and using multiple key shares at each share server is not addressed in [8]; (b) the security analysis in [8] is ad hoc and very preliminary; (c) the ARECA system organization and its security features are not mentioned in [8]; (d) the implementation details, the testing results, and the performance analysis we presented in this paper are not covered by [8]. This paper includes substantial extensions to [8].

8. CONCLUSION

This paper presents ARECA, a highly attack resilient CA. Compared with existing threshold-cryptography-based intrusion tolerant CA system, the uniqueness of ARECA is that it engineers a novel two phase signature composition scheme and a multi-layer CA protection architecture. As a result, ARECA is (a) practical, (b) highly resilient to both insider and outsider attacks that comprise one or more components, and (c) can prevent a variety of outside attacks. We have implemented the ARECA prototype. And the performance evaluation results are very encouraging.

Besides analyzing the resilience of ARECA when multiple combiners are compromised, there are several issues which we want to investigate in the near future. For example, how to update key shares on both share servers and combiners, how to detect compromised share servers, how to detect compromised combiners, how to reconfigure ARECA when there is a compromised component, and how to add a new share server or combiner.

ACKNOWLEDGMENTS

We thank the members of the work group in China, which include Dr. Liwu Zhang, Jing Yu, Zheng Cao, and several other researchers for developing and testing the prototype. We thank Qing Liu for editing the paper. The work is supported by the National 863 program of China under award 2002AA141060 and award 2003AA144050. Peng Liu is supported in part by DARPA and AFRL, AFMC, USAF, under award F20602-02-1-0216 and by NSF CCR-TC-0233324.

9. REFERENCES

[1] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, Moti Yung, "Optimal-Resilience Proactive Public-Key Cryptosystems", *Proc IEEE Symposium on Foundations of Computer Science*, Miami Beach, Florida , 1997:384-393

[2] T. Wu, M. Malkin, D. Boneh, "Building Intrusion Tolerant Applications", In *Proc USENIX Security Symposium*, Aug 1999, pages 79-91.

[3] V. Shoup, "Practical Threshold Signatures", in Proc *Enrocrypt 2000*, Belgium, pages 207-220

[4] P. S. Gemmell, "An Introduction to Threshold Cryptography", in *CryptoBytes*, a technical newsletter of RSA, 1977, Vol. 2, No 7:7-12

[5] L. Zhou, F. B. Schneider, R. V. Renesse, "COCA: A Secure On-line Certification Authority", *ACM Transactions on Computer Systems*, 20(4): 329-368, 2000

[6] A. Shamir, "How to share a secret", Communications of the ACM, 22: 612-613, 1979

[7] D. Boneh, G. Durfee, and Y. Frankel, "Exposing an RSA Private Key Given a Small Fraction of its Bits", In K. Ohta and D. Pei, editors, *Advances in Cryptology- Asiacrypt 1998*, number 1514 in Lecture Notes in Computer Science, pages 233-260. Springer Verlag, 1998

[8] Jiwu Jing, Dengguo Feng, "An Intrusion tolerant CA scheme", *Chinese Journal of Software*, 13(8), 1417-1422, 2002