http://www.zju.edu.cn/jzus; E-mail: jzus@zju.edu.cn
ISSN 1009-3095 *Journal of Zhejiang University SCIENCE* V.4, No.5, P.555-559, Sep.-Oct., 2003

# Distributed certification application via a trusted dealer*

LIU Duan-yang(刘端阳)†, PAN Xue-zeng(潘雪增), PING Ling-di(平玲娣)

(*Department of Computer, Zhejiang University, Hangzhou 310027, China*)

†E-mail: ldy_zj@hotmail.com

Received Sept.10, 2002; revision accepted Jan.2, 2003

**Abstract:** Distributed certification via threshold cryptography is much more secure than other ways to protect certification authority (CA)'s private key, and can tolerate some intrusions. As the original system such as ITTC, etc., is unsafe, inefficient and impracitcal in actual network environment, this paper brings up a new distributed certification scheme, which although it generates key shares concentratively, it updates key shares distributedly, and so, avoids single-point failure like ITTC. It not only enhances robustness with Feldman verification and SSL protocol, but can also change the threshold $(t, k)$ flexibly and robustly, and so, is much more practical. In this work, the authors implement the prototype system of the new scheme and test and analyze its performance.

**Key words:** Distributed certification, Trusted dealer, Robust, Flexible
**Document Code:** A **CLC number:** TP393.08

## INTRODUCTION

It is well-known that certification authority (CA) is the core component of public cryptographic infrastructure (PKI); and that the security of its private key is very vital. If the private key is compromised, the whole PKI will be corrupted. Generally, there are two types of CA: On-line and Off-line. Off-line CA is more secure, but not good for PKI's development. On-line CA can provide easy and quick services; but needs measures to secure its private key and keep its services stable.

Secret sharing via threshold cryptography can tolerate internal or external intrusions and will promote the security. Generally, Lagrange secret sharing in Shamir(1979) is used; but it requires reconstruction of the key in a place to sign a message, and is easy to fail. ITTC system in Malkin *et al*. (2000) presented a distributed certification scheme, which can tolerate intrusions and certify a request without reconstruction of a private key.

But ITTC system does not satisfy practical application completely. Although it generates RSA key pairs distributedly and avoid single-point failure, it needs the assumption that each entity in secret sharing is absolutely honest and reliable, which is very difficult to satisfy in practice. And the generation also takes too much time, which is decribed in Malkin's experiment. With increasing key size, time expense will double even much more. At the same time, the share distribution is not robust and secure. The scheme in Frankel *et al*.(1998) is more robust, but requires much more time than that of ITTC system.

In this paper, we bring up a new scheme wherein key generation centers on a trusted dealer, but key update is distributive; and so avoid single-point failure. At the same time, the new scheme enhances robustness with Feldman verification mechnisms in key generation and update. Moreover, it can change the threshold value $(t, k)$ flexibly and robustly, which is required in practice.

The paper describes the main process and algorithm of the new scheme, and their implementation. Section 2 describes the system model. Section 3 and 4 demonstrates algorithms of the

* Project supported by the Nature Science Foundation of China(No. 69974031) and Zhejiang Nature Science Foundation (No. 600014)

key operation. Section 5 presents the detailed performance of the whole system.

## SYSTEM MODEL

The whole system is shown in Fig. 1. Each share server in Fig. 1 administrates his own key share. Share servers are servers of CAs in the sense of functionality.

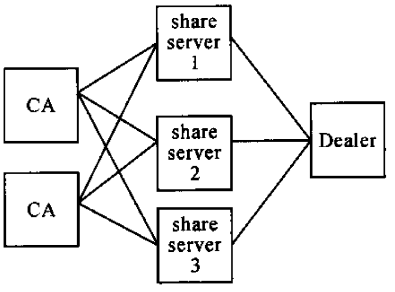


**Fig.1 System model**

CA: CA in Fig. 1 is different from general certificate authority and is a client relative to share servers. CA interacts with outside users and receives their certification request. After CA receives the certification request, it distributes this request to $t$-out-of-$k$ share servers to sign. After receiving the reply from the share servers, CA multiplies the results and makes a check (detailed check protocol in Malkin *et al*., 2000) before responding to the certification request. CA and share servers can be viewed as a whole, and for certification applicants the whole process is transparent just as normal CA. That is to say, an outsider communicating with the CA is unaware that the corresponding private key is stored in shared form. The detailed process of distributed certification is given below:

Setup: Let threshold cryptography be $t$-out-of-$k$, let $d_i$ be the share key of the share server and $N$ be the modulus of RSA;

Step 1: CA server receives the certification request, which is in PKCS10 (Public Key Cryptography Standards # 10) format. According to the request and system policy, CA server generates the corresponding certificate message $M$ and computes the hash value $H = hash(M)$ (Note: SHA-1 algorithm is used in this system).

Step 2: CA server randomly selects the trusted set $S \subset \{1, 2, \cdots, k\}$ where $|S| = t$, and distributes the hash value $H$ and the set $S$ to these $t$ selected share servers (Note: each peer identifies the others with SSL protocol and inner X.509v3 certificates).

Step 3: Share server $i$ signs the hash value $H$ with his share $d_i$ and computes the signature $SIG_i \equiv H^x \bmod N$ where $x = d_i \cdot \prod_{\nu \in S-\{i\}} (i - \nu)^{-1}(0 - \nu)$. Then each responds with corresponding result.

Step 4: CA server collects $t$ share servers' signatures $SIG_i$, and computes $SIG \equiv H^A \prod_{i=1}^{t} SIG_i \bmod N$. Then the signature $SIG$ is verified with RSA public key. If the verification fails, key check protocol will be applied to identify the corrupted shares. CA server will kick the corrupted share out, reselect the set $S$ and return to step 2. (Note: key check protocol is described in Malkin's article).

Step 5: CA server generates X.509v3 with the signature $SIG$ and the message $M$, and thus accomplishes the certification.

Note: The certificates of inner entities can pledge to accomplish the computation of the value $x = d_i \cdot \prod_{\nu \in S-\{i\}} (i - \nu)^{-1}(0 - \nu)$. Each share server can know its value of from the CN field of its certificate (see the description of the entity of Dealer). And no one can change the value of $i$.

**Share server:** CA's private key is shared with threshold cryptography. Each share server administrates his share key. Of course, a share server can manage multiple keys and serve a number of CAs. As private key is shared with threshold cryptography, the whole system can tolerate intrusion in some degree. Each server holds one share which leaks nothing about the private key. So, even if a few share servers are penetrated and the secrets stored in them are exposed or corrupted, there is no compromise to the overall system safety. In other words, an attacker learns nothing from penetrating a few of the share servers. The system can identify corrupt share servers and recover compromised keys by key shares update protocol.

**Dealer:** In the whole system, the dealer is a trusted base, and can issue certificates for inner entities, generate RSA keys, update key shares and other functions.

1) Inner certificate issuance: The whole sys-

tem uses the Secure Socket Layer(SSL) to authenticate and encrypt all communication. The dealer governs the issuance of a X.509v3 certificate to each entity(CA, share servers and dealer itself ). In order to identify each entity, the CN (Common Name) field(RFC 2459) in each certificate contains a string that allows other parties to authenticate it. The field (CN) is described in the following table:

**Table Common name of each entity**

| Entity | CN |
|---|---|
| CA | [CERAUTH N] |
| Share Server | [SHARE N] |
| Dealer | [DEALER N] |

When establishing an SSL-secured connection, both peers must send their certificates to establish mutual authentication. Each peer verifies the certificates it receives and then parses it to extract the identifying string to ensure that the other peer is authentic. If a certificate is not sent, if verification fails, or if the identity string does not match what was sent in the protocol, the connection fails. At the same time, the certificates of share server show its value $i$ and no one can change it. So each share can compute its share of certification signature correctly and avoids the need for key reconstruction in a place.

2) Key shares generation: On his initiative, the dealer can generate RSA key pairs locally, compute the $t$-out-of-$k$ shares of private key and distributes key shares to the corresponding share servers. During the process of share distribution, Feldman verification can ensure shares are correct and any forged and compromised share of the private key will be kicked out. The detailed discussion is in Section 3.

3) Key shares update: Although CA's private key is secure via threshold cryptography, as a mobile adversary who may penetrate different share servers in different time period, it is not secure enough. A mobile adversary may get the private key by penetrating $t$ (system threshold) share servers; so shares must be updated periodically, after which the new share of each share server is different from the old one. And no adversary can get any information about new shares from the old shares. The dealer will instruct all share servers to update its shares by key shares update protocol (described in Section 4).

## KEY SHARES GENERATION

The normal secret sharing is Shamir's Lagrange Algorithm (Shamir, 1979 ), which is efficient but cannot secure shares distribution. Distributive key shares generation more secure than Shamir's, such as Boneh and Franklin's algorithm (Boneh *et al*., 1997) and the algorithm of Frankel *et al*. (1998). Boneh and Franklin's avoids the failure of single point, is not secure and efficient enough. BF algorithm requires that every share server is honest but curious, which the practical network environment does not satisfy. From the test result of the Malkin *et al*. (1999) algorithm, it is not very efficient. Sometimes it needs 20 minutes to generate RSA keys. The Frankel *et al*. (1998) algorithm is more secure and robust than the BF algorithm, but needs a large amount of computation and takes much more time and network bandwidth.

We bring up a new key generation method based on a trusted dealer. Key shares are generated in the trusted dealer and distributed securely and robustly by Feldman verification and SSL protocol. Although key shares are generated in a place initailly, it can update key shares distributedly (described in Section 4), which also avoids the possibility of single-point failure. The key shares generation is described below:

Setup: Trusted dealer $D$ generates RSA parameters: a modulus $N$, a public key $e$ and a private key $d$. Let $K = 11!$ and the threshold be $(t, k)$, where $k \geqslant 2t - 1$ and $k < 11$. As gcd $(e, K^2) = 1$, $A$, $B$ can be computed to satisfy $1 = eA + K^2B$, using the extended Euclidean algorithm. Note that $d \equiv A + d'L^2 \bmod \Phi(n)$ where $d' \equiv dB \bmod \Phi(n)$.

Step 1: $D$ chooses a random polynomial: $F(x) = F_0 + F_1 x + \cdots + F_{t-1} x^{t-1}$, where $F_j \in_R \{0, K, \cdots, 2K^3 N^{2+e} t\}$ and $F(0) = F_0 = d'L^2$ for $1 \leqslant j \leqslant t - 1$. $D$ computes shares: $F(i)$ for $1 \leqslant i \leqslant k$ and distributes them securely by SSL protocol.

Step 2: $D$ randomly chooses $g \in Z_n^*$, where $g$'s order is maximal, and computes Feldman

verifications values: $E_i \equiv g^{F_i} \bmod N$ for $i \in \{0, \cdots, t-1\}$ and publishes $g$ and all Feldman verification values.

Step 3: Share server $i$ verifies the value $F(i)$ whether

$$g^{F(i)} \equiv \prod_{j=0}^{t-1} (E_j)^{\nu} \bmod N$$

where $V = i^j$ or not;

Step 4: If all share servers pass the verification, the corresponding share key $d_i = F(i)$. Or else, the process restarts from Step 1;

Step 5: $D$ computes each public key share $V_i \equiv g^{d_i} \bmod N$ and destroys secrets : the private key $d$, polynomial $F(i)$ and its coefficients.

## KEY SHARES UPDATE

In order to prevent a mobile adversary who can corrupt different share servers in different time period, we need update all key shares periodically and recover corrupted shares. The protocol below is not only able to update and recover shares, but is also able to change the threshold $(t, k)$ flexibly and robustly. Although key share is generated in one place in this scheme, key update protocol is distributive and able to avoid single-point failure. The detailed protocol is descibled below:

Step 1: Dealer $D$ randomly selects a trusted set $S \subset \{1, 2, \cdots, k\}$ where $|S| = t$. And $D$ generates a random message $M$, and demands signature of this message from $t$ selected Share Servers.

Step 2: $D$ verifies the signature with the public key. If it is not correct, the protocol reselects the trusted set $S$ and restarts from step 1.

Step 3: $D$ instructs the $t$ selected Share Servers to update and others to wait.

Step 4: Share Server $i \in S$ randomly constructs $t$-degree polynomial:

$$F_i(x) = F_{i,0} + F_{i,1}x + \cdots + F_{i,t-1}x^{t-1}$$

where its coffients $F_{i,j}$ belong to $\{0, K, \cdots, 2K^3N^{2+e}t\}$ for $j$ belong to the set $\{1, \cdots, t-1\}$ and $F_i(0) = F_{i,0} = d_iZ_i$ where $Z_i = \prod_{j \in S-\{i\}} \frac{-j}{i-j}$. Share Server $i$ computes $d_{i,j} = F_i(j)$ for $j$ belongs to the set $\{1, \cdots, k\}$ and Feldman verification values $E_{i,j} = g^{F_{i,j}}$ for $j \in \{0, \cdots, t-1\}$. Then it distributes $d_{i,j}$ securely by SSL protocol and publishes all verification values $E_{i,j}$.

Step 5: Share server $j(j \in \{1, \cdots, k\})$ receives its secret share $d_{i,j}$ and all verification values $E_{i,j}$ and verifies each share like Feldman's article. If the verifications is correct, it computes new secret share $d_j = \sum_{i \in S} d_{i,j}$ and computes and publishes the public key $V_j = g^{d_j}$.

Note: In order to agree with practice, the threshold $t$ and $k$ can be changed in step 3, which the dealer $D$ can instruct share servers with a new threshold $(t, k)$.

## PERFORMANCE

The system prototype is implemented on open SSL(2001.11). The SSL protocol provides mutual authentication with X.509v3 certificates. The system makes a standard extension in the format of X.509v3, which gives a uniform sequence number to each share server. And so no one can imitate share servers. At the same time, in order to recover corrupted shares, the system satisfies $k \geqslant 2t-1$, which can tolerate some intrusions. The following timing measurements of distributed certification were taken on six intel-based PC's. All were running on Redhat Linux (release 7.0). The computers were connected by 10base-T Ethernet.

Latency and throughput are the parameters of characterization of the system. Latency is the time that a CA has to wait for all requests to be serviced. And throughput is the number of requests that can be serviced per second. Table 1 shows the latency and throughput for several key-sizes. And Table 2 shows the performance of two CA servers with 10 tasks and 1024 bits key, which are runing at the same time.

We can learn something regular from Table 1 and Table 2 that the latency and throughput have something to do with key-sizes, task numbers and the value of $t$ and $k$. A key of larger size results in more communications and longer computing time, so as key size increases, the throughput decreases and the latency increases. And the more tasks, the higher loads on the servers and the latency increases. But key shar-

ing can balance the loads in some degree as different share servers can be selected at concurrent requests, so on the opposition the throughput increases. The balancing ability of key sharing is related to the ratio $t/k$, and the smaller this ratio is, the more requests can be processed. At the same time, key sharing tolerates $t-1$ restricted servers being corrupted and distributed certification can continue to serve efficiently.

**Table 1 Latency and throughput**

| key size | tasks | no sharing | | 2-out-of-3 | | 3-out-of-5 | |
|---|---|---|---|---|---|---|---|
| 1024bits | 2 | 0.044s | 45.7/s | 0.276s | 7.24/s | 0.371s | 5.38/s |
| 1024bits | 10 | 0.226s | 44.2/s | 1.046s | 9.56/s | 1.263s | 7.92/s |
| 2048bits | 2 | 0.234s | 8.54/s | 0.623s | 3.21/s | 0.730s | 2.74/s |

**Table 2 Multi CA servers**

| CA | 2-out-of-3 | | 3-out-of-5 | |
|---|---|---|---|---|
| CA0 | 1.753s | 5.705/s | 1.852s | 5.399/s |
| CA1 | 2.121s | 4.715/s | 2.341s | 4.272/s |

## CONCLUSIONS

Threshold cryptography can secure CA's private key much more than general methods. This paper describes in detail a new scheme for distributed certification via threshold crytography and solves the vital problems of distributed certification such as key generation, key update, etc.

## References


Boneh, D. and Franklin, M., 1997, Efficient Generation of Shared RSA Keys. *In*: Proceedings Crypto'97, Springer Press, California, p.425 – 439.

Feldman, P., 1987. A Practical Scheme for Noninteractive Verifiable Secret Sharing. *In*: Proceedings of the 28th Symposium on Foundations of Computer Science, IEEE Computer Society Press, New York, p. 427 – 437.

Frankel, Y., MacKenzie, P. D. and Yung, M., 1998. Robust Efficient Distributed RSA-Key Generation. *In*: Proceeding of the thirtieth Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, p.663 – 672.

Malkin, M., Wu, T. and Boneh, D., 1999. Experimenting with Shared RSA Key Generation. *In*: Proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS), Springer Press, California, p. 43 – 56.

Malkin, M., Wu, T. and Boneh, D., 2000. Building Intrusion Tolerant Applications. *In*: Proceeding of DARPA Information Survivability Conference and Exposition, IEEE Computer Society Press, New York, **1**:74 – 87.

Open Security Socket Layer (OpenSSL), 2001. Available at http://www.open-ssl.org/

Public Key Cryptography Standard (PKCS), 2001. Available at http://www.rsa-security.com/rsalabs/pkcs/

Shamir, A., 1979. How to share a secret. *Communications of ACM*, **22**(11):612 – 613.

Request For Comment 2459 (RFC 2459), 2002. Available at http://www.ietf.org/rfc/