ISSN 1009 – 3095 *Journal of Zhejiang University SCIENCE* V.4, No.3, P.287 – 293, May – June, 2003
http://www.zju.edu.cn/jzus; http://www.periodicals.com.cn; jzus@zju.edu.cn

Building a highly available and intrusion tolerant database security and protection system (DSPS)*

CAI Liang(蔡亮)†, YANG Xiao-hu(杨小虎), DONG Jin-xiang(董金祥)
(*Institute of Artificial Intelligence, Zhejiang University, Hangzhou 310027, China*)
†E-mail: cail2000@21cn.com

Received June 12, 2002; revision accepted Oct.10, 2002

Abstract: Database Security and Protection System (DSPS) is a security platform for fighting malicious DBMS. The security and performance are critical to DSPS. The authors suggested a key management scheme by combining the server group structure to improve availability and the key distribution structure needed by proactive security. This paper detailed the implementation of proactive security in DSPS. After thorough performance analysis, the authors concluded that the performance difference between the replicated mechanism and proactive mechanism becomes smaller and smaller with increasing number of concurrent connections; and that proactive security is very useful and practical for large, critical applications.

Key words: Information warfare, Proactive security, Intrusion tolerant, DSPS(Database Security and Protection System)
Document code: A **CLC number:** TP309.2

INTRODUCTION

How to protect the database, the kernel resources of information warfare, is becoming more and more important since the rapid development of computer and communication technology. The Artificial Intelligence Institute of Zhejiang University has prototyped a Database Security and Protection System (DSPS) to counter the possible malicious behavior of untrusted DBMS. Based on the malicious DBMS threat model (Cai *et al.*, 2002a), DSPS can greatly improve the critical application's capability to fight malicious DBMS by incorporating self-controlled authentication, principal/object mapping, transparent attribute encryption/decryption, attribute/tuple level mandatory access control, and parallel structure of multiple DBMSs (Cai *et al.*, 2002b).

Proactive security is a new security technology developed in recent years. It combines threshold cryptography and key refreshment protocol to improve the security of the critical system keys during its whole life cycle (Canetti *et al.*, 1997). Compared with the ordinary security technology, proactive security does not reactive only after the system has been intruded and the critical keys have been exposed, but periodically executes the key refreshment protocol to fight the possible undetected attacks.

The authors incorporate proactive security technology to construct the DSPS server group. The coherence of the two server group structures (the DSPS server group structure to improve the availability and the group structure needed by proactive security to distribute the cryptographic key) enable implementation of a highly available, intrusion tolerant DSPS server group using less hardware resources.

RELATED RESEARCHES ON PROACTIVE RSA

Proactive security combines the approach calling for distribution of trust with the one of periodic refreshment: Proactive = Distributed + Refresh (Canetti *et al.*, 1997). That is, first distribute the cryptographic capabilities among several servers. Next, have the servers periodically implement a refreshment protocol that allows servers to automatically recover from possible, undetected break-ins, and in particular pro-

* Project (No.45.6.1-017) supported by the National Defense Pre-Research Fund of China

vides servers with the new shares of the sensitive data while keeping the sensitive data unmodified. Very importantly, information gathered by an attacker before a refreshment period becomes useless for attacking the system in the future. In all, the security of the system will be guaranteed as long as not too many of the servers are broken into between consecutive implementations of the refreshment protocol.

Now we show how a private RSA key can be broken up into a number of pieces (shares). Each share can be stored on a separated key server and yet the private key can be used without having to reconstruct the secret (Malkin *et al*., 2000). The basic idea is to pick random integers d_1, d_2, d_3 so that $d_1 + d_2 + d_3 = d$; then store share d_i on key server i, for $i = 1, 2, 3$. When a client wishes to apply d to sign a message M it sends M to all three key servers. Each server applies its own share di to obtain $S_i = M^{d_i}$, and sends the result S_i back to the client, who obtains S_1, S_2, and S_3 from the three servers; and multiplies them to obtain the signature $S = S_1 \times S_2 \times S_3$. Since $d = d_1 + d_2 + d_3$, we have $S = M^d$ as required.

Clearly this approach generalizes to distributing a private RSA key among k servers, called k-out-of-k sharing. But there are a number of problems with the k-out-of-k sharing mode. Most importantly, it is not fault-tolerant. If one of the share servers crashes, the entire system goes offline. If one of the share servers loses its share, the private key is lost forever. For this reason, t-out-of-k sharing is more usable. Any t of the share servers can be used to apply the key.

We are aware of three implementation efforts of proactive security systems. The Ω system (Meiter *et al*., 1996) built at AT&T also uses threshold cryptography to protect private keys; but it does not support detection of corrupted servers or the ability to refresh shares in case a share server is compromised.

The proactive security toolkit built at IBM focuses on using proactive security applied to DSS (Barak *et al*., 1999). DSS and RSA have different sharing properties. RSA keys are easy to share, but are hard to generate distributively. On the other hand, DSS keys are easy to generate distributively, but are hard to use for threshold signatures (Malkin *et al*., 1999). That is why the DSPS prototype uses proactive RSA. In DSPS's deployment environment the RSA keys are generated by trusted sever locally, not distributively.

Stanford University's Intrusion Tolerant via Threshold Cryptography (ITTC) project (Malkin *et al*., 2000) intended to embed the proactive security support into existing applications (such as certificate authority and Apache web server) is also based on proactive RSA, but engages too many key servers. In this paper we make use of the topological coherence of replicated structure and proactive share servers, trying to implement a practical, not expensive critical application environment.

HIGHLY AVAILABLE AND INTRUSION TOLERANT DSPS

In order to improve the availability and the system throughput, and also to reduce the average response time, several DSPS servers provide services to clients as a group. The communication between client and DSPS server is secured by SSL protocol, using RSA algorithm. In this deployment, how to protect the private key of the DSPS becomes the security kernel. As shown in Fig. 1, there are two traditional approaches to protect the private key d:

(1) Replicated: Each server in the server group stores a copy of the private key d. This is the fastest approach to apply the d because the encryption and decryption can be performed locally in each server. Besides, the private key d will not be lost unless all the servers are corrupted. But this approach is very sensitive to intrusions. The private key d will be lost if attackers break into any of these servers.

(2) Centralized: There is a specialized Key Server responsible for storing and applying the private key d. All the DSPS servers must ask the Key Server to perform the cryptographic operation related to d. In this approach compromising the DSPS servers will not reveal d. The key is exposed only if the Key Server is compromised. In addition, the key is lost forever if this Key Server loses it. In addition, system performance is rather low because all the cryptographic opera-

tions (CPU intensive) are actually performed by a single machine.

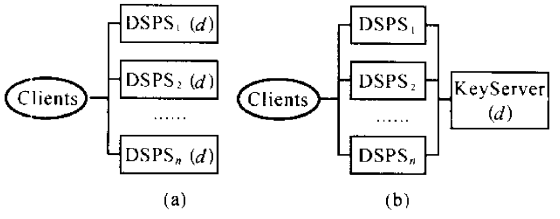

Fig.1 Traditional approaches to protect the key d
(a) Replicated Approach; (b) Centralized Approach

Deployment structure of DSPS

As a security system designed for preventing information warfare attacks, DSPS has two special requirements:

(1) Everything possible should be done to prevent attacks from succeeding. But most importantly, it should also be assumed that not all attacks will be averted at the outset, thus placing increased emphasis on the ability to live through and recover from successful attacks.

(2) In order to reduce the cost and time of deploying DSPS it can use the certificates issued by the existing PKI, if there is one. More importantly, if there is no PKI, DSPS could issue the certificates itself so as to reduce the dependence on other systems. DSPS would incorporate some functions of certificate authority and issue the certificates to all the clients (and the possible subordinate certificate authorities), and the clients would save the public key of DSPS for later authentication and session key generation. Since there are many clients in the typical DSPS deployment environment, the cost of replacing the public key in all clients is very expensive once the private key of DSPS is exposed.

So we incorporate the proactive security technology to construct DSPS server group. The coherence of the two server group structures (the DSPS server-group structure to improve the availability and the group structure needed by proactive security to distribute the cryptographic key) enable implementation of a highly available, intrusion tolerant DSPS server group using less hardware resources. The final DSPS deployment structure is shown in Fig.2.

Local Proxy is responsible for intercepting all requests from applications to DBMS and forwarding them to appropriate DSPS i using random algorithm. If the selected DSPS i goes offline because of being attacked or other reasons Local Proxy will try to find another available DSPS server. In order to assure the usability of the DSPS system we do not modify any applications. Local Proxy will monitor the original DBMS service port, and receive the original request from applications.

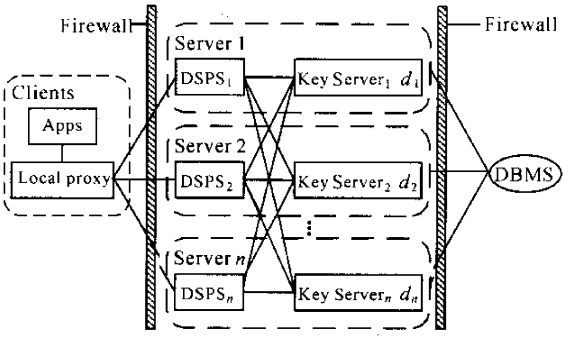

Fig.2 Highly available and intrusion tolerant DSPS deployment structure

KeyServer running like a daemon on the server machine. Each KeyServer$_i$ holds a share d_i of the DSPS private key d and all the KeyServers form the server group for storing and applying the d. Each KeyServer can serve multiple DSPS servers concurrently. Clearly it is desirable that an adversary is not able to compromise the KeyServers. However, the intrusion tolerant design of DSPS's deployment structure ensures that even if a few KeyServers are penetrated and the shares stored on them are exposed or corrupted, overall system security is not compromised. In other words, an attacker learns nothing from penetrating KeyServers less than threshold. The system identifies corrupt KeyServers and can be instructed to take appropriate action to refresh the shares stored on them.

DSPS is the main security monitor responsible for enforcing self-controlled authentication, principal mapping, object mapping, mandatory access control, and the like. It uses the cryptographic function provided by KeyServer group to finish the authentication, session key generation and message signature generation. When a DSPS server connects to the KeyServers it first authenticates itself as an authorized client (Each DSPS i has its own certificate.). It then interacts with

the KeyServers to sign a message or decrypt a given ciphertext using the shared private key stored in the KeyServer group.

Comparison of three different key management mechanisms is shown in Table 1.

Table 1 Comparison of three key management mechanisms

	Replicated	Centralized	Proactive
Fault tolerance	Good	Bad	Good
Intrusion tolerance	Bad	Moderate	Good
Performance	Good	Bad	Moderate
Coherence with DSPS	Good	Bad	Good

Next, we will use the 3-out-of-4 sharing as an example to illustrate the system initialization, applying the shared private key, detection of compromised KeyServers, and the key refreshment protocol.

System initialization

The cryptographic initialization routine locally generates the modulus N, public exponent e and private exponent d. Then it picks random integers d_1, d_2, d_4, d_5 and computes $d_3 = d - d_1 - d_2$, $d_6 = d - d_4 - d_5$. According to the share allocation table shown in Table 2, each KeyServer holds multiple d_i's as indicated by the column corresponding to that KeyServer. Observe that in the allocation table, any three servers can apply the key while any two servers learn nothing about d. The combinatorial construction of the t-out-of-k share allocation table is discussed in (Malkin *et al*., 1999).

Table 2 Share allocation table of 3-out-of-4 sharing

$d = d_1 + d_2 + d_3 = d_4 + d_5 + d_6$

KeyServer_1	KeyServer_2	KeyServer_3	KeyServer_4
d_1	d_2	d_3	d_3
d_4	d_4	d_5	d_6

In order to strengthen system security, the KeyServer does not save the shares in its local storage. Instead, they are stored in four IC cards. Shares d_1 and d_4 are encrypted by the administrator's passphrase into IC_1, d_2 and d_4 into IC_2, d_3 and d_5 into IC_3, d_3 and d_6 into IC_4. Besides the key shares, each IC card stores the modulus N and the share allocation table. Every time when the system starts, KeyServer i requires the insertion of IC_i, then decrypts the key shares, modulus, and share allocation table into memory using the administrator's passphrase.

Now the key initialization process is finished. The public key of DSPS can be copied as needed, but from then on the private key will only exist as key shares.

Applying the shared private key

We are now ready to describe the example interaction that takes place when a client requests the KeyServers to sign a message M using the private key d (The encryption, decryption, and verification processes are similar). Each DSPS_i keeps a table KeyServer.Status[1..4] indicating the status of all KeyServers. The value of KeyServer.Status[i] can be NORMAL, OFFLINE, or COMPROMISED, meaning the KeyServer_i is working normally, currently offline or compromised. Initially the DSPS_i set all the KeyServer.Status to NORMAL.

When signing a message, DSPS_i picks a random coalition of three KeyServers that are NORMAL (Suppose they are KeyServer_1, KeyServer_2, and KeyServer_4.), then send the following message to each of the servers in the coalition:

DSPS_i	$\text{DSPS}_i.\text{KeyServer}_j.\text{Seq}$	SIGN	S_1	S_2	S_4	M

where DSPS_i is the sender's identifier, $\text{DSPS}_i.\text{KeyServer}_j.\text{Seq}$ is the request sequence number, and SIGN is the command code.

After receiving the request, KeyServer_j first checks the validity of the sequence number $\text{DSPS}_i.\text{KeyServer}_j.\text{Seq}$ to prevent some replay attacks. Then based on the coalition being used and the share allocation table read from IC_j during startups, the KeyServer_j extracts appropriate d_j (KeyServer_1 use d_1, KeyServer_2 use d_2, KeyServer_4 use d_3), and returns to DSPS_i.

DSPS_i collects all three responses from KeyServer_1, KeyServer_2, KeyServer_4 and computes the required signature $S = S_1 \times S_2 \times S_4 \bmod N = M^d \bmod N$. If DSPS_i cannot connect to KeyServer_j, or cannot receive the response from KeyServer_j in the specific time interval, DSPS_i alters the KeyServer.Status[j] to OFFLINE and the message signing process is restarted. Note

that $DSPS_i$ cannot simply save the partial results from other KeyServers, send the same request to another NORMAL KeyServer, and collect the response as a substitute for different coalition results in different share used by each $KeyServer_j$ to compute the partial result S_j.

Then $DSPS_i$ will verify the message signing process using the public key e. If $S^e = \pm M \bmod N$ the signature is correct and the process terminates. Otherwise, for each server in the coalition, $DSPS_i$ performs a zero-knowledge test to validate the server's response. The test is described in the next subsection. All servers that send incorrect values are marked as COMPROMISED and the message signing process is restarted.

Throughout the interaction the status information in KeyServer.Status is displayed in the DSPS system's monitor board that helps the administrator understand the current system status and take appropriate countermeasures. If the number of KeyServers which are NORMAL is less than the threshold 3, the system cannot provide services to clients. If the number of compromised KeyServers is more than 3, the system is already compromised and the private key d is exposed; and the whole cryptographic system must be reinitialized.

Identification of compromised KeyServer

In the above subsection, if $DSPS_i$ fails to verify the signature there is at least one KeyServer which is compromised. In order to identify this compromised KeyServer we perform a zero-knowledge test.

(1) We first add the following attributes into the public key: a random integer g, a public key share $V_i = g^{d_i}$ corresponding to each private share d_i. Suppose the correct response from $KeyServer_j$ is S_j, $S_j = M^{d_j} \bmod N$.

(2) Challenge: $DSPS_i$ picks random integers a and b and computes $Z = H(M^a g^b \bmod N)$. H is a cryptographic hash function. We use MD5. Next, send Z to each server in the selected coalition.

(3) Verification: The legitimate server will respond with $A_j = Z^{d_j} \bmod N$. Now $DSPS_i$ can check if $A_j = H(S_j^a V_j^b \bmod N)$; If not, $KeyServer_j$ is considered to be compromised.

The above protocol is a simplification of a protocol from (Malkin *et al.*, 2000). We only substitute MD5 for SHA-1. According to Malkin *et al.*(2000), with Small Order Assumption, a KeyServer which does not have the right d_j will fool the protocol with probability at most $1/\sqrt{N}$ (For a 2048 bit key, the probability is less than $1/2^{1024}$). Further more, if the hash function used is a random function, the protocol is zero-knowledge.

Key refreshment protocol

The major characteristic of proactive security is invoking the refreshment protocol periodically. If the administrator refreshes before the adversary can break into a threshold of machines, the adversary gains nothing. Second, refresh can be used to generate new shares to replace lost or corrupted shares. The key refreshment protocol is relatively mature. Currently we incorporate a key refreshment protocol from (Malkin *et al.*, 2000). Limited by the paper length, we do not introduce here, interested users please refer to (Malkin *et al.*, 2000).

PERFORMANCE ANALYSIS

To make clear the performance difference caused by these key management mechanisms, our test focused on the performance of private key operations in a large number of simulated concurrent connections. The test was performed on an isolated 10M shared Ethernet. A Pentium III 550, 128M RAM PC was used as the client, generating lots of threads sending requests to the DSPS servers. Another four Celeron 500 128M RAM PCs were used as the server machines, running DSPS servers and KeyServers. All these PCs ran Linux operating system.

We focused on the average response time and system throughput. The test results of 1024 bit RSA algorithms are shown in Fig.3. The average response time increased with the increasing number of concurrent connections, but the system throughput showed jittery response. Careful analysis led to the conclusion that it is caused by repeatedly thread switching under large number of concurrent connections. Aside from the impact of thread switching, the system bottleneck is the CPU capacity because there are a large number of digital signing operations which are very CPU

intensive. So whatever the key management mechanism selected, the more CPUs engaged in the cryptographic operations, the larger throughput can be attained. Therefore, the centralized mechanism has the smallest throughput (and the biggest average response time) because there is only one server performing the cryptographic operations. On the contrary, the replicated mechanism results in the biggest throughput and the smallest average response time (using the same number of CPUs). The performance of the proactive mechanism is between that of the centralized mechanism and the replicated mechanism.

Next, we try to analysis how much performance degradation is caused by the proactive mechanism, compared with replicated mechanism. Limited by the paper length, we only present the comparison of average response time between the replicated mechanism and proactive mechanism of 1024 bit RSA in Table 3. We can see that the performance difference becomes smaller and smaller with increasing number of concurrent connections. In very large number of connections, the proactive mechanism even surpasses the replicated one (probably caused by uncertain switching of context). Considering that only few private key operations are needed in connecting a client and that there are relatively few connections in typical DBMS applications, the proactive security is very useful and practical in large critical applications.

Table 3 Average response time of different key management mechanisms for 1024 bit RSA (in second)

Concurrent Connection	50	100	150	200	250	300	350	400	450	500
Replicated(3 Server)	0.407	1.284	2.51	3.779	5.609	7.913	10.058	13.612	16.693	18.286
Replicated(4 Server)	0.332	0.78	1.463	2.285	3.404	4.485	5.681	7.538	8.611	10.448
2-out-of-3 sharing	0.761	1.762	3.119	4.466	6.313	8.193	10.029	11.904	13.221	15.491
2-out-of-4 sharing	0.6	1.346	2.104	3.021	4.206	5.306	6.602	7.965	9.123	10.285
3-out-of-4 sharing	1.141	2.222	4.572	5.714	6.779	9.011	10.446	11.892	14.312	16.462
Centralized (3 Server)	1.575	3.831	8.391	9.98	16.79	22.332	24.88	26.148	29.056	31.39
Centralized (4 Server)	1.842	4.124	8.039	17.375	19.896	23.513	27.282	28.166	36.765	40.482

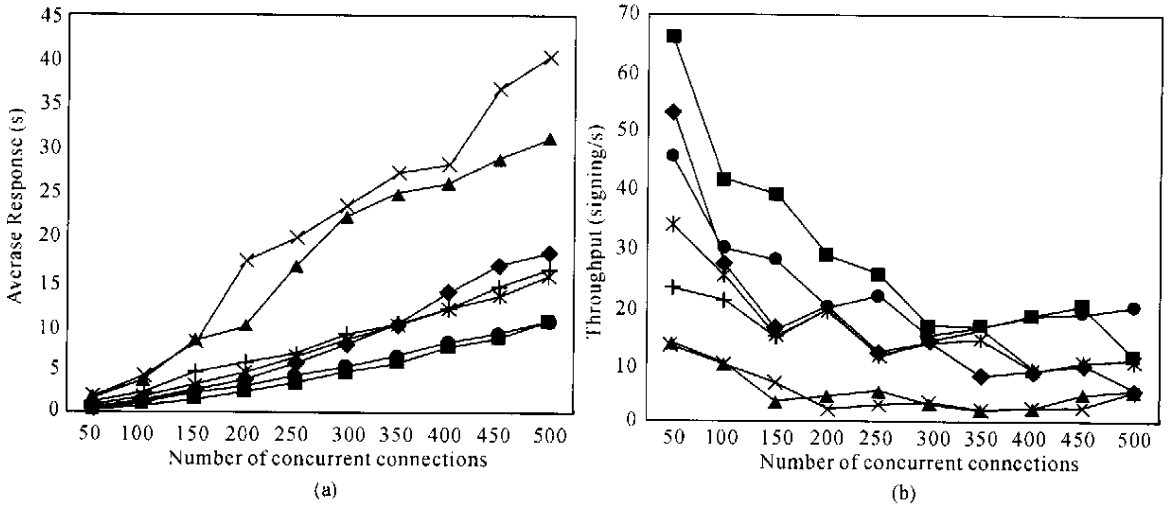

Fig.3 Comparison of three different key management mechanisms for 1024 bit RSA
(a) Average response; (b) Throughput

CONCLUSIONS

Considering the two special requirements of DSPS (designed for preventing information warfare attacks and the ability to be deployed without the support of underlying public key infrastructure), we incorporated proactive security technology to construct the DSPS server group. The coherence of the two server group structures (the DSPS server-group structure to improve the availability and the group structure needed by proactive security to distribute the cryptographic key) enable implementation of a highly available, intrusion tolerant DSPS server group using less hardware resources.

The authors' detailed performance analysis of three different key management mechanisms led to the conclusion that the performance difference between replicated mechanism and proactive mechanism becomes smaller and smaller with increasing number of concurrent connections. As only few private key operations are needed to connect a client, and as there are relatively few connections in typical DBMS applications, proactive security is very useful and practical for large critical applications.

References

Barak, B., Herzberg, A., Naor, D. and Shai, E., 1999. The proactivesecurity toolkit and applications. Proceedings of the ACM Conference on Computer and Communications Security, ACM, Singapore, p.18 – 27.

Cai, L., Yang, X.H. and Dong, J.X., 2002a. Database securityin information warfare - special requirements and antagonism in China. *Journal of Computer Research and Development*, **39**(5):568 – 573(in Chinese).

Cai, L., Yang, X.H and Dong, J.X., 2002b. A referencemodel for database security proxy. *Journal of Zhejiang University SCIENCE*, **3**(1):30 – 36.

Canetti, R., Gennaro, R., Herzberg, A. and Naor, D., 1997. Proactivesecurity: long-term protection against break-ins. *CryptoBytes: the technical newsletter of RSA Labs*, **3**(1):1 – 8.

Malkin, M., Wu, T. and Boneh, D., 1999. Experimenting withshared generation of RSA keys. Proceedings of the Internet Society's Symposium on Network and Distributed System Security, IEEE Computer Society Press, California, p.43 – 56.

Malkin, M., Wu, T. and Boneh, D., 2000. Building intrusion tolerant applications. Proceedings of DARPA Information Survivability Conference & Exposition (DISCEX), IEEE Computer Society Press, California, p.74 – 87.

Meiter, M., Franklin, M., Lacy, J. and Wright, R., 1996. The Ω keymanagement service. Proceedings of the ACM Conference on Computer and Communications Security, ACM, New Delphi, India, p.38 – 47.