A Novel Approach Share Key Refreshing for Long Term Protection in Distributed Cryptography by Proactive Security

Rajkumari Retoliya , Prof. Anshu Tripathi
Department of Information Technology ,Mahakal Institute of Technology
Ujjain, INDIA

***Abstract*— Security is an important issue for networks, especially for those security-sensitive applications. In the entire environment of security, it is necessary to ensure privacy. This Paper provides share key refreshing technique for maintaining the security of the system, even when some nodes are in control of attacker. It provides an automated recovery of the security of individual components, avoiding the use of expensive and inconvenient manual processes. The technique can be used with threshold cryptography, by providing periodic refreshments of the sensitive data held by the servers. This way, the proactive approach guarantees uninterrupted security as long as not too many servers are broken into at the same time. In this paper a solution is provided to enhance security among communication channel. Towards this a share key refreshing technique is represented in the form of algorithm in which some trusted nodes share the private key, which is a part of private-public key pair. It is distributed on some trusted systems according to threshold cryptography. Now once shares are distributed they must be refreshed otherwise attacker can easily obtain those shares and generate the key. Once shares are distributed they must be refreshed after some period of time. We also present implementation details for this scheme.**

***Keywords*—** *Threshold Cryptography, Proactive Security, Attacks, Secret sharing, Distributed Cryptography, Share Refreshment.*

I. INTRODUCTION

Security is an important issue for networks, especially for those security-sensitive applications. In the entire environment of security, it is necessary to ensure privacy. Everyone in the group must be aware of the security goals and to be conscious in achieving them [1]. As data communication is becoming more pervasive, complex and the use of digital data becoming much more widespread, data security has become a wider, more complex and more important problem. Cryptography can be an important tool to help in improving security. Since public key cryptography and threshold cryptography is widely used now a day for security purpose but still there is a problem arises regarding the safety of the private key. There is no secure mechanism still be achieved in order to protect the private key from the attacker [2]. For this proactive share key refreshing technique provides a mechanism, in which the shares are again divided into sub shares and transferred to each other for updating the old shares. In such a way attacker cannot easily obtain all the shares at the same time before refreshment. Proactive security (PC) combines the ideas of distributed cryptography with the refreshment of secrets. In this approach, shares are periodically renewed in such a way that information gained by the attacker in one time periods is useless for attacking the secret after the shared are refreshed. These are the core properties of the *Proactive Security*. It doesn't wait until a break-in is detected. Instead, it invokes the refreshment periodically in order to maintain uninterrupted security or force detection [3]. This novel approach ensures secrecy and authenticity of communication, with automated refresh of the secret keys.

Cryptography offers a set of sophisticated security tools for a variety of problems, from protecting data secrecy, through authenticating information and parties, to more complex multi-party security goals. Yet, the most common attacks on cryptographic security mechanisms are system attacks. Such system attacks are done by intruders (hackers, or through software trapdoors using viruses or Trojan horses), or by corrupted insiders. Unfortunately, such attacks are very common and frequently quite easy to perform, especially since many existing environments and operating systems are insecure (in particular Windows).

As a result, computer and network security involve a set of tools to prevent and detect intrusions, and to regain control over a computer from the attacker. Detection is particularly important, since once an attack is detected on any one computer system administrators is alarmed and are likely to regain control from the attacker- on most or all computers. Furthermore security measures are likely to be tightened, and at least some security exposures found and fixed. Therefore, attackers often do their best to avoid detection, and indeed often give up control over a computer rather than risk being detected [3]. During this time when the Internet provides essential communication between tens of millions of people and is being increasingly used as a tool for commerce, security becomes a tremendously important issue to deal with. There are many aspects to security and many applications, ranging from secure commerce and payments to private communication and protecting passwords [4]. One essential aspect for secure communications is to protect private key using proactive security from adversary which is the focus of this paper.

II. KEY SECURITY ISSUES

As our world is growing increasingly dependent on digital systems, security of these systems is becoming increasingly critical. In addition to accidental failures, threats of malicious attacks must be addressed by the security systems of today and tomorrow. Connectivity of the digital systems has become an integral part of their functionality. However, connectivity could also provide malicious attackers with an easy access to the system, in particular allowing them to mount their attacks even from the other side of the globe. Physical isolation is hardly ever an option in achieving protection, and so most systems must rely on other mechanisms for their security. These mechanisms, be they simple passwords authentication or sophisticated cryptographic tools, generally depend on maintaining some secrets keys. Thus, security of a system hinges on the condition that the attackers cannot gain access to its secret keys. This condition may be difficult to satisfy, especially since these keys must be actively used by the system. One might try to make it harder for an adversary to expose the secret keys. To this end one might utilize special devices (such as smart-cards), multiple factor mechanisms (e.g., regular passwords, combined with smart-cards, and biometric mechanisms), etc. But our experience shows that no matter how strong is the protection of the secret keys, it is very likely that a sufficiently motivated adversary will succeed sooner or later and expose these keys. Thus, an experienced security systems designer will plan explicitly for the event of key exposures.

A.Key Attacks

The problem with keys in general is that there are so many ways to get at them. Types of attacks on the key are:

1.Wireless Attacks: Most of the things that have been said so far about protecting keys apply regardless of the type of security system we are using. They are not specific to wireless. Wireless, of course, introduces a whole new set of opportunities for attackers trying to get keys because it is so easy to access the data streams, even though they may be encrypted. The problem for the attacker is that the data is encrypted and he/she needs the keys. Assuming we don't change the keys, he/she has as much time as he/she wants to capture sample messages and analyze them.

2. Brute force attacks: The brute force method means that an attacker tries every possible key until he finds a match. Given that he knows the ciphertext and protocol, he would start with a key value of all zeros, decrypt the message, and see whether it matches the plaintext (or any fragments he has). If he keeps adding 1 to the key value, in principle, he will sooner or later hit on the right key because all possible keys will have been tried. The time taken for a brute force attack depends on the key size, or more correctly the key entropy. This is one of the reasons that government export controls tend to be set according to *key length*. For example, it used to be that you could not export any security technology from the United States with a key length of more than around 40 bits. Because the 40-bit key is crackable, many security systems use larger keys. The use of a longer key really renders brute force attacks completely ineffective, assuming the underlying cryptographic algorithm has no weaknesses. Let's suppose supercomputers become faster and we can try a hundred keys in a microsecond. With a 104-bit key, you would still need (on average) 3,200,000 billion years to find the right key.

3. Dictionary Attacks: Given that we can so easily defeat brute force attacks by adding a few bits to the key, any attacker with an IQ in the double digits will look for another approach. Here's the idea: Instead of trying every possible key, try only those keys that we think the user is likely to use. For example, the attacker could assume that the key is made up entirely of letters and numbers, as is typical for user-chosen passwords. This reduces key entropy. A 104-bit key is now only as effective as a 78-bit key because only 6 bits of every byte are used. However, 78 bits is still uncrackable using brute force so the attacker must narrow down further. This approach to reducing the number of keys to test brings us to the idea behind a dictionary attack. In a dictionary attack, the enemy uses a huge dictionary, or database, containing all the likely passwords. This will certainly include every word in the English language and may contain other languages as well.

4. Algorithmic Attacks: If the enemy cannot mount a brute force or a dictionary attack, another approach is to try to break the algorithm that is, to try to find a flaw in the way the encryption is performed that might expose the key value. It is difficult to describe these algorithmic attacks generally because they depend so much on the algorithm and understanding the weaknesses often requires that we are a cryptographic expert. However, there is a straightforward analogy with safe breaking.

B.Key Security Goals

The goals for a security system design can be formulated as threefold:

(a) Make key exposures as difficult and as expensive for adversaries as possible,

(b) if/when the keys are exposed, minimize the damage;

(c) Recover from the exposures.

III. Distributed Cryptography

In general, it is not convenient that the security of a system relies on the behavior of a single agent. Let us consider, for instance, the case of a certification authority, a trusted entity that certifies that a given public key corresponds to a given user. Clearly, a certification authority that is composed by several independent servers is more reliable than one that is formed by a single server.

Distributed cryptography, introduced in 1987, makes it possible to design cryptographic systems in which some operations require the collaboration of several users. Concretely, a distributed cryptosystem is a public key cryptosystem in which the secret key is shared among a set of users. Only some qualified subsets of users will be able to perform the operation related to the secret key (decrypting or signing). In this way, the security of the system is increased, because the loss or theft of several shares of the secret key does not necessarily break the system's security. Several distributed cryptosystems have been proposed until now. Most of them have a threshold structure, that is, the sets of users that are able to execute the protocol are those having a certain number of elements. Due to this fact, distributed cryptography is called also in general threshold cryptography. Distributed cryptography is currently a very active research field that is related to many different areas in cryptology. There are several open problems whose solution would lead to the construction of more efficient and versatile distributed cryptosystems. Many of these problems are related to the different cryptographic protocols that are used as pieces of a distributed cryptosystem[3].

IV. Threshold Cryptography

Threshold cryptography is the technique of distribution of trust. Threshold Cryptography is based on (n, t) scheme where n is number of server and threshold t is the number of node which recovers the original message, whereas it is infeasible for at most (n-t) servers to recover the original message. In Threshold Cryptography, private key (k) is distributed to n number of nodes, assign one share to each server. Each server is known as a shareholder. Each shareholder generates a partial signature using own partial private key. Combiner is a node which verifies the trust of the servers according to the protocols defined on them. It receives the partial signature generated by all the servers, while accepting only those partial signatures generated by trustworthy servers. With t correct partial signatures, the combiner is able to compute the signature for the certificate. However, compromised servers cannot generate correctly signed certificate by themselves, because they can generate at most (n-t) partial signatures. A combiner can verify the validity of a computed signature. In case verification fails, the combiner tries another set of partial signatures [11].

A Problems with Threshold Crptography

When we use public key cryptography technique, once the keys are distributed then there is no mechanism to keep them secure against attacks. So if once attackers manage to attack on the system then he can easily get the private key. But now we have so many technologies like threshold cryptography which uses the distributed key method where key is not kept on a single system but one has to divide the private key into sub parts and then distribute trust/private key share (one from key pair) among trusted nodes. But still they face a problem that once private key shares are distributed then they cannot be refreshed because there is no previous mechanism to do this. Because of this the probability of compromising trusted nodes and getting the shares of the key increases, which in turn can help the attacker to get the whole private key of his need.

In threshold cryptography the private key is divided using some algorithm like RSA, ELGAMAL etc., and distributed between the trusted servers. These servers do not have the whole private key, but the part of the key. So it is difficult for the attacker to obtain the key. But the problem arises when the attacker is able to gain all the t shares from (n, t) threshold scheme by attacking on all the trusted nodes. This work is moved around the solution of this problem[12].

B Proposed Approach

The threshold cryptography gives a way to convey a shared key to a node without using any key infrastructure also is very suitable for a secret sharing in network. However, given t or more shares in an (n, t) threshold cryptography scheme the secret S can be found. Without the share refresh and with finite span of time it is not very hard for malicious nodes to compromise at least t share holder nodes and finally obtain the secret key. To make each share refresh without disclosure of any share or a secret key itself. Proactive secret sharing can be employed. It allows refreshing all shares by generating a new set of shares for the same secret key from the old shares without reconstructing the secret key. In the proposed work the same approach has been implemented.

V. Proactive Security

Proactive security is a mechanism for protecting against such long-term attacks. It combines the approach calling for distribution of trust with the one of periodic refreshment:

Proactive = Distributed + Refresh.

That is, first distribute the cryptographic capabilities among several servers. Next, have the servers periodically engage in a refreshment protocol. This protocol will allow servers to automatically recover from possible, undetected break-ins, and in particular will provide the servers with new shares of the sensitive data while keeping the sensitive data unmodified [3]. Proactive security shows how to maintain the overall security of a system even under such conditions. In particular it provides automated recovery of the security of individual components, avoiding the use of expensive and inconvenient manual processes. The technique combines two well-known approaches to enhance the security of the system: distributed (or threshold) cryptograph, which ensures security as long as a threshold (say half) of the servers are not corrupted; and periodic refresh (or update) of the sensitive data (e.g. keys) held by the servers. This way, the proactive approach guarantees uninterrupted security as long as not too many servers are broken into at the same time. Furthermore, it does not require identification when a system is broken into, or after the attacker loses control; instead, the system proactively invokes recovery procedures every so often, hoping to restore security to components over which the attacker lost control[8]. The main new contributions (assumptions) are:

- A secure initialization mechanism, with reasonable, practical requirements from the computer and operating system. Specifically, all we require is a secure boot process (which is a good idea anyway, against viruses - and easily done with signed code); and a per-machine secret-private key pair, with the public key protected from modification (e.g. in ROM or write-once EEROM), and the secret key in erasable memory (e.g. disk). Previous results required storage of parameters specific to the particular application (such as the group's public key) in secure storage, which is not practical.
- A set of application program interfaces (APIs) that allow the use of the toolkit to improve security, specifically provide security in spite of break-ins into computers, of existing applications, as well as the development of new applications which are proactive secure.

The security of any proactive solution relies heavily upon its correct architecture and integration with existing, non-proactive, operating system. The design of system, which does not view the proactive model as series of protocols but, rather, as a security enhancement of the operating system which transforms it into a proactively secured system via the appropriate use of proactive protocols, has not been defined nor implemented in the past. We show that it is possible to transform general applications which are required to remain secure for long periods of time to operate in a proactive environment, namely proactivizing applications [14].

To this end, we define architecture for a proactive operating environment which serves as a platform on which standard applications can be proactivized. This operating environment consists of a network of servers which are set up once, which we call the proactive network.

Each server is instantiated at boot time by the operating system and is checked periodically, also by the operating system. Servers can recover data (both public and private data) from other servers in the proactive network, if such data is corrupted or lost. Once the proactive network is set up, any application can run on the top of the network and request proactive services by the means of API.

VI. Share Refreshment

A. Share Refreshment methodology

Proactive schemes are proposed as a countermeasure to mobile adversaries. A proactive threshold cryptography scheme uses share refreshing, which enables servers to compute new shares from old ones in collaboration without disclosing the service private key to any server. The new shares constitute a new (n, t + 1) sharing of the service private key. After refreshing, servers remove the old shares and use the new ones to generate partial signatures. Because the new shares are independent of the old ones, the adversary cannot combine old shares with new shares to recover the private key of the service[7].Thus, the adversary is challenged to compromise t + 1 servers between periodic refreshing. Share refreshing relies on the following homomorphic property. If (s1 1, s12, . . . , s1n) is an (n, t + 1) sharing of k1 and (s21, s22, . . . , s2n) is an (n, t + 1) sharing of k2, then (s11 + s21, s12 + s22, . . . , s1n + s2n)v is an (n, t + 1) sharing of k1 + k2. If k2 is 0, then we get a new (n, t + 1) sharing of k1.

Given n servers. Let (s1, s2, . . . , sn) be an (n, t + 1) sharing of the private key k of the service, with server i having si. Assuming all servers are correct, share refreshing proceeds as follows: first, each server randomly generates (si1, si2, . . . , sin), an (n, t+1) sharing of 0. We call these newly generated sij 's subshares. Then, every subshare sij is distributed to server j through a secure link. When server j gets the subshares s1j , s2j , . . ., snj, it can compute a new share from these subshares and its old share (s0j = sj + _n i=1sij). Share refreshing must tolerate missing subshares and erroneous subshares from compromised servers. A compromised server may not send any subshares.However, as long as correct servers agree on the set of subshares to use, they can generate new shares using only subshares generated from t + 1 servers. For servers to detect incorrect subshares, we use verifiable secret sharing schemes. A variation of share refreshing also allows the key management service to change its configuration from (n, t+1) to (n0, t0 +1). This way, the key management service can adapt itself, on the fly, to changes in the network: if a compromised server is detected, the service should exclude the compromised server and refresh the exposed share; if a server is no longer available or if a new server is added, the service should change its configuration accordingly. The essence of the proposed solution is again share refreshing. The only difference is that now the original set of servers generate and distribute subshares based on the new configuration of the service: for a set of t + 1 of the n old servers, each server i in this set computes an (n0, t0 + 1) sharing (si1, si2, . . . , sin0) of its share si and distribute subshares sij secretly to the jth server of the n0 new servers. Each new server can then compute the new share from these subshares. These new shares will constitute an (n0, t0 + 1) sharing of the same service private key. More generically we can say that refreshments of the shares can be performed as follows:Each server i chooses a random t-degree polynomial fi(X) such that fi(0)=0. Server i then sends to server j the value sij=fi(j) mod q. still Server j then computes its new refreshed share s'j as follows: s'j = sj + s1j + ... + snj mod q and erases its old share. The new shares s'i lie on the polynomial f'(X) = f(X) + f1(X) + ... + fn(X) which is of degree t and whose free term is still s so the new shares define the same secret. Recovery of the share sr=f(r) mod q of server r is also possible as follows [16]. Each server i chooses a random t degree polynomial gi(X) such that gi(r) =0 (by choosing the free coefficient of gi(X) to satisfy this condition). Server i then sends to server j the value gij=gi (j) mod q. Server j then computes its new share g'j : g'j = g1j + ... + gnj mod p. The new shares g'i lie on the polynomial g'(X) = g1(X) + ... + gn(X) which is still of degree t and satisfies g'(r) = 0. Now, each server i sends to server r the value si+g'i mod q = f(i)+g'i mod q. Server r interpolates these values to construct the polynomial f(X) + g'(X) from which it derives f(r)+g'(r) mod q and this is exactly sr=f(r) mod q, its lost share[2].

C.Share Refreshment Algorithm

Key Generation:

1. Randomly choose 2 prime numbers p and q

Compute a(n) = (p-1) * (q-1) (i)

Detemine server's private key (d) as

$$d = e^{-1} \bmod a(n) \qquad \text{(ii)}$$

Where e is server's public key

2. Determine the threshold (t) as

$$t >= n\backslash 2 \qquad \text{(iii)}$$

So that t < n and n should be greater than or equal to 2.

Where n is number of trusted nodes

Evaluate the Shares (Key distribution):

3. Share generation is based on Shamir's scheme

Secret is a value S in the set of integers [0……….p-1]

Where p is prime number.

Server (who is sharing the secret) generates t random numbers ($a_1 \ldots a_t$) and put these values into given the polynomial.

$$f(X) = (S + a_1X + \ldots\ldots + a_tX^t) \qquad \text{(iv)}$$

4. If f(X) is calculated for different nodes i.e. node 1, node 2, node 3 Where X = 1, 2, 3………respectively.
Each node receives own share.
Share Refreshment:
Each share holder randomly generates own sub-shares (e.g. ($s_{i1,}$ $s_{i2\ldots}$ s_{in}) on node i), and each subshare is mutually exchanged to refresh own share. Explained as below:
5. Let ($s_1, s_2, \ldots, s_n$) be an (n, t) sharing of the secret key S of the service, with node i having $s_{i.}$
6. Node i (i Є {1 . . . n}) randomly generates s_i's subshares (s_{i1}, $s_{i2}, \ldots, s_{in}$) for an (n,t) sharing of 0.
7. Every sub-share s_{ij} (j Є {1 . . . n}) is distributed to node j through secure link.
8. When node j gets the sub-shares (s_{1j}, s_{2j}. . . s_{nj}), it computes a new share from these subshares and its old share with an equation:

$$s'_j = s_j + \Sigma^n_{i=1} s_{ij} \qquad (v)$$

9. Now each share (s'_1, s'_2, ……. s'_n) is an (n, t) sharing of the secret key S, because

$$\Sigma^n_{j=1} s_{ij} = 0 \qquad (vi)$$

After each proactive secret sharing, all nodes will change (refreshed), so that old shares become useless. In such case, since it is impossible to obtain new shares from old share, a malicious node must collect at least shares before the refreshment of share which makes his job difficult.

VII. Implementaion Details

The Proactive Security software has been prototyped in Java 1.6.

A. Architectural Flow

The project consists of various modules. At the very first step it is necessary to distribute the trust or the secret key to the trusted nodes or machines using (n,t+1) threshold cryptography. Suppose we have N trusted machines then there should be n shares of the key. In the figure5.1, it is shown that key is distributed among nodes using (n,t+1) threshold cryptography. These shares are distributed among trusted nodes (machines), which are the parts of the private key whose public key is distributed among all nodes. We are assuming that keys are already distributed among trusted nodes. After this my work has been started, in the next step, we have to generate a mechanism by which shares can be divided into subshares. Now all nodes share their subshares between themselves. They send their subshares through a secure communication channel, which can be obtained by using some cryptographic algorithm. By using this subshare mechanism, the shares are divided into subshares. Because of this we can refresh the shares of the nodes. And now if attacker wants to achieve all parts of the secret then he should have to attack on each and every node but at the same time the share refreshing mechanism works and refreshes the shares. Now attacker cannot achieve the whole shares because old shares are renewed before he got all shares. In this way this mechanism provides an efficient security. If any node becomes compromised by the attacker then that node can also be recovered. The compromised node can be detected by its behavior. So in this way we can obtain a more secure communication channel and it can be use for a long term.

B. Implementation Issues

The proactive environment architecture and its algorithms constitute quite a complex system to implement and test. As such, the Java language was a natural choice for implementation since it provides a fast and simple prototyping environment. Moreover, its portability across platforms was an important feature, since different nodes in the proactive network may have to run the toolkit on entirely different platforms (for example, our demo runs on a network of five nodes, which are Windows based). Yet, this choice of programming language had a number of implications.
1. Erasing information from memory, which is an absolutely necessity for the correct implementation of secrets refresh, is an issue in all environments (due to virtual memory) and, in particular, in a garbage collected environment like Java, since garbage collectors typically copy memory as part of the collection process.
2. The code for a Java program includes the code for the JVM (Java Virtual Machine), as well as the byte-code of all classes loaded (dynamically) by the machine in the course of its execution. Therefore, satisfying the code-validation assumption for Java programs may require assistance from the JVM, possibly by using mechanisms like signed classes or by writing a customized class loader.
3. We were able to use some of the more advanced features of Java to simplify both the protocols and communication. All protocols and messages are implemented as subclasses of an abstract superclass.
In this way all protocols are treated in a uniform way, which simplifies both the dispatch of messages to protocols and the addition of new protocols. In addition, we didn't have to define 'protocol messages' in a strict, well structured, way and parse them. Instead, all messages are sent as serialization of some object.

C. API Implementation

Using the Java language enabled us to implement the API between the server and its clients in a convenient and simple way. To write such an application firstly we have to write a class name as StartNodeServer for which we have to import various packages of java like:

1) import java.math.BigInteger;
2) import java.net.*;
3) import java.util.Random;
4) import java.io.*;

ne more public class is created named ApplicationConstants in which all the port numbers are defined, for establishing communication between servers. This a common class called in all the other classes. Three more servers (trusted nodes) are defined, which takes part in share refreshing. These servers are those machines who shares the parts of the private key named as Node1Server, Node2Server and Node3Server.

For all these three machines we have three classes named Node1Server, Node2Server and Node3Server. The packages which are importing here as follows:
1) import java.io.DataInputStream;
2) import java.io.DataOutputStream;
3) import java.io.IOException;
4) import java.io.InputStream;
5) import java.io.OutputStream;
6) import java.math.BigInteger;
7) import java.net.ServerSocket;
8) import java.net.Socket;
9) import java.net.SocketTimeoutException;

To write a proactive application the client must write a class which is a subclass of the class **ApplicationConstatnts.** This superclass provides its subclass with methods to request services from the server, send messages to clients running on the other machines, and load new classes to the server.
In addition, this class defines abstract methods which the subclass must implement and which the server uses to notify the client about the status of request and about incoming messages from other clients.

ACKNOWLEDGMENT

We would like to thank the all faculty members of the institute, who helped us lot in calculating the facts and figures related to our paper and the anonymous reviewers who provided helpful feedback on my manuscript.

REFERENCES

[1] G. R. Blakley, "Safeguarding cryptographic keys". In Proc. AFIPS 1979 National Computer Conference, pp. 313-317. AFIPS, 1979.
[2] D. Boneh and M. Franklin. "Efficient generation of shared RSA keys". In Proc. Crypto '97, pp. 425-539.
[3] R. Canetti, R. Gennaro, A. Herzberg and D. Naor, "Proactive Security: Long-term protection against break-ins". CryptoBytes: the technical newsletter of RSA Labs, Vol. 3, number 1 - Spring, 1997.
[4] Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. "Optimal resilience proactive public-key cryptosystems". In Proc. 38th Annual Symp. on Foundations of Computer Science. IEEE, 1997.
[5] Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. Proactive RSA. In Proc. of Crypto '97. 12. P. Gemmell. "An introduction to threshold cryptography". In Cryptobytes, Winter 97, pp. 7-12, 1997.
[6] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin, "Robust threshold DSS signature". In Ueli Maurer, editor, Advances in Cryptology - Eurocrypt '96, pp. 354-371, 1996. Springer-Verlag Lecture Notes in Computer Science No. 1070.
[7].Stanislaw Jarecki and Nitesh Saxena. Further simplifications in proactive RSA signature schemes. In LNCS, volume 3378, pages 510–528, 2005. TCC'05.
[8]. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk and M. Yung. "Proactive public key and signature systems", ACM Security '97.
[9]. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing, or: How to cope with perpetual leakage". In D. Coopersmith, editor, Advances in Cryptology – Crypto '95, pp. 339-352, 1995. Lecture Notes in Computer Science No. 963.
[10]. T. Rabin, "A simplified approach to threshold and proactive RSA", Proc. of Crypto '98.
[11]. A. Shamir. How to Share a Secret. Communications of the ACM, 22:612-613, 1979.
[12].Timo Warns. "On the Coverage of Proactive Security: An Addition to the Taxonomy of Faults" Lecture Notes in Informatics, 67 . Gesellschaft für Informatik e.V., pp. 405-409. ISBN 3-88579-396-2,2005.
[13]. S. Jarecki, N. Saxena, and J. H. Yi. An Attack on the Proactive RSA Signature Scheme in the URSA Ad Hoc Network Access Control Protocol. In *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, pages 1–9, October 2004.
[14]. B. Barak, A. Herzberg, D. Naor, and E. Shai. The Proactive Security Toolkit and Applications. In Proc. 6th ACM Conference on Computer and Communications Security (CCS). ACM, 1999.
[15]. M. Jakobsson, S. Jarecki, H. Krawczyk, and Moti Yung. "proactive RSA for constant-size thresholds". Unpublished manuscript, 1995.
[16]. http://www.research.sun.com/projects/crypto.
[17]. http://www.rsa.com/rsalabs/.
[18]. L. Ertaul and N. Chavan, "Security of Ad Hoc Networks and Threshold Cryptography", in MOBIWAC 2005.