International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 6 – No.2, September 2013 – www.ijais.org

Implementation of Shamir's Secret Sharing on Proactive Network

Saria Islam
Senior Lecturer
Department of Computer Science and Engineering
IBAIS University, Dhaka, Bangladesh

A. S. M Mahmudul Hasan
Lecturer
Department of Computer Science and Engineering
Hamdard University Bangladesh, Narayangang

ABSTRACT

For most cryptosystems, by using a single-system master key there is a need to protect many important encryption and decryption keys used to achieve data security. There are three major drawbacks under this single-master key arrangement. First, if the master key is disclosed to the public by accident, then the entire system has no secrecy at all. Second, if the master key is lost, then all the keys under protection become inaccessible. Third, if the owner of the master key becomes disloyal, then all important information becomes completely available to the opponents. The secret sharing scheme was designed to overcome these problems.

Shamir's (t, n)-threshold scheme is one of the most well-known examples of secret sharing schemes which provides a very simple and efficient way to share a secret among any t of the n participants.

The primary goal of our research is to study the terminology and protocols behind secret sharing schemes and implement Shamir's scheme using Java. In addition, we have also introduced a new scheme for extending the Shamir's scheme on proactive network.

General Terms

Cryptography and Network security.

Keywords

Cryptography, Secret Sharing, Shamir's secret sharing scheme, Proactive Secret Sharing, Implementation of Shamir's Secret Sharing.

1. INTRODUCTION

The question whom to trust is fundamental for anybody in any situation, and becomes paramount when security is required. There are many examples when even well established trusted entities become malicious. The reasons can be different, but the result at the end is always the same: they are not trusted anymore. One known solution to overcome such a problem is instead of placing your trust just in one (trusted) party to distribute the trust among a group of entities, especially when the stakes are high. This should be done so that certain specified groups of them are able to perform an operation, but smaller, possibly malicious, subsets cannot do any harm to the system. There are several known cryptographic concepts that address the question for distribution of trust. Some of them are secret sharing schemes, verifiable secret sharing schemes, and multiparty computation [1].

Although a well-known term within the cryptographic community, secret sharing might be a bit misleading for an outsider [2]. It does not mean two or more people sharing one secret. It means two or more people are having shares of the secret. One secret is split into n different shares. Only when merging at least t, $0<t\leq n$, number of shares the initial secret can be reconstructed. Ideally, it should be impossible to gain any information about the secret with less than n shares. In their work Practical Cryptography, Ferguson and Schneier mention that the value of secret sharing techniques in reality is very limited. The arguments are that it is complex to operate, and that most companies do not have a group of responsible people who distrust each other. However, there is another side of it too. The main purpose of dual combination locks on bank vaults is not to prevent the employers from taking cash, but to stop their families from being taken hostage [3]. The employer's inability to open the lock on her own removes the inducement for criminals to force her. Secret sharing schemes can be used in equivalent situations, or perhaps to safely increase data availability, just to name two examples. Secret sharing is not only an interesting information-theoretical concept, but it also has several practical applications.

A well-known principle in the analog world is the term reduced trust, meaning that in order to keep a secret, the less knowledge or power each entity has, the better. This is the basic philosophy, and we shall study how it is implemented in the digital world as well.

Consider the following problems:

- In some situations, there is usually one secret key that provides access to many important files. If such a key is lost (e.g., the person who knows the key becomes unavailable, or the computer which stores the key is destroyed), then all the important files become inaccessible. The question one may ask is how to back up secret information, so that it does not depend on one authority only.
- While performing the encryption procedure, a certain key needs to be stored; as we want to ensure that no single entity is entrusted with too much knowledge or power, the question now, is how to ensure that the key will not be exploited by the authority holding it.

2. SECRET SHARING

Suppose you and your friend accidentally discovered a map that you believe would lead you to an island full of treasure. You and your friend are very excited and would like to go home and get ready for the exciting journey to the great fortune. Now who is going to keep the map? Suppose you and your so-called friend do not really trust each other and are afraid that, if the other one has the map, he/she might just go alone and take everything. Now we need a scheme that could make sure that the map is shared in a way so that no one would be left out in this trip. What would you suggest?

An easy way to solve this problem is to split the map into two pieces and make sure that both pieces are needed in order to find the island. Now we give one piece to each. You can happily go home and be assured that your friend has to go with you in order to find the island. This illustrates the basic concept of secret sharing [4].

In cryptography, secret sharing refers to any method for distributing a secret among a group of participants, each of which allocates a share of the secret. The secret can only be reconstructed when the shares are combined together; individual shares are of no use on their own [5].

3. SHAMIR'S (t, n) - THRESHOLD SCHEME

Shamir's secret sharing scheme is a threshold scheme based on polynomial interpolation [6]. It allows a dealer D to distribute a secret value s to n players, such that at least players are required to reconstruct the secret. The protocol is information theoretically secure, i.e., any fewer than t players cannot gain any information about the secret by themselves [7].

3.1 The Sharing Protocol

Please use a 9-point Times Roman font, or other Roman font with serifs, as close as possible in appearance to Times Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

Goal: To share the secret *s* among players P_1, P_2, ……, P_n such that *t* players are required to reconstruct the secret.

1. Dealer *D* creates a random polynomial *f*(*x*) of degree *t*-1 and constant term *s*.
   $$f(x) = a_0 + a_1x + \blacksquare + a_{t-1}x^{t-1}$$
   This polynomial is constructed over a finite field, such that the coefficient a_0 is the secret *s* and all other coefficients are random elements in the field; the field is known to all participants.

2. Dealer *D* publicly chooses *n* random distinct evaluation points: $X_j \neq \square 0$, and secretly distributes to each player P_j the share
   $$share_j(s) = \left(X_j, f(X_j)\right)$$
   $$j = 1 \ldots\ldots.. n$$

(**Remark:** The evaluation point X_j could be any publicly known value, therefore for our convenience, we assume $X_j = j$, hence the shares are denoted as$f(1), \ldots, f(j), \ldots, f(n)$.

3.2 The Reconstruction Protocol

Goal: To reconstruct the secret from each subset of *t* shares out of *n* shares. Without loss of generality we will mark this subset: $f(1), \ldots, f(t)$

1. Use Lagrange interpolation to find the unique polynomial *f*(*x*) such that$\deg f(X) < t$ and $f(j) = share_j(s)$ for *j=1,2,..t*
2. Reconstruct the secret to be *f(0)*.

Interpolation Property: Given t pairs of *(i,f(i))*, with *i*'s all distinct, there is a unique polynomial *f(X)* of degree *t-1*, passing through all the points. This polynomial can be effectively computed from the pairs *(i,f(i))*.

Lagrange interpolation: $f(x) = \sum_{i=1}^{t} f(i) \times L_i(X)$ where $L_i(x)$ is the Lagrange polynomial:

$L_i(X) = \frac{\prod_{j\neq 1}(x-x_j)}{\prod_{j\neq 1}(x_i-x_j)}$ which has value 1 at X_i, and 0 at every other X_j.

Note: in the following sections the terms shareholder and server can be interchanged.

Graphic-representation of a degree-2 polynomial and its shares is shown in figure..

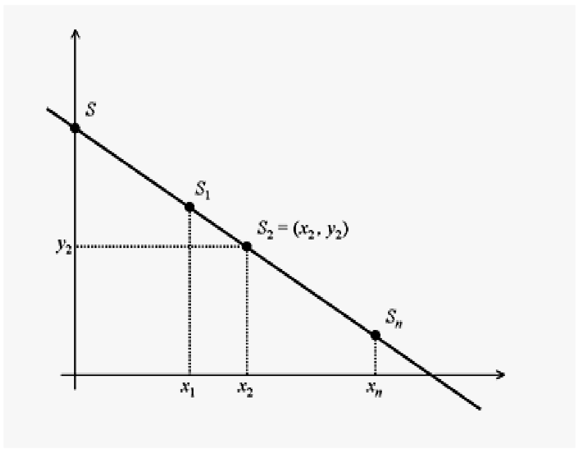


Fig 1(a): Shamir's Secret Sharing Scheme

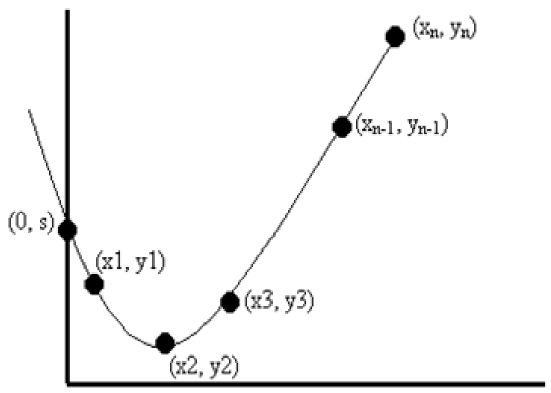


Fig 1(b): Shamir's Secret Sharing Scheme

4. PROACTIVE SECRET SHARING

We need a scheme that allows servers to generate a new set of shares for the same secret from the old shares without reconstructing the secret. 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 re-cover 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. Very importantly, information gathered by an attacker before a refreshment period becomes useless to attack the system in the future.

Such a scheme is called a proactive secret scheme (PSS). We have argued that PSS is needed for server recovery. But, in reality, break-ins to a server are very hard to detect, especially when the attacker simply steals certain secret information without modifying anything on the victim server. An attacker can cover his tracks when he exits. To strengthen the security of a replicated service, we can invoke our PSS periodically (at regular intervals). (see Figure below)

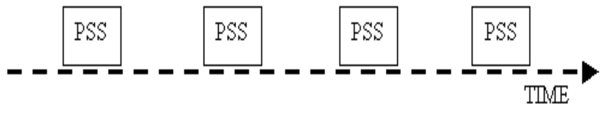


Fig 2: Proactive Secret Sharing

Before the execution of the PSS, every server checks the integrity of its code and state, trying to remove any attackers that might exist in that server at that point in time. How would our PSS improve security through periodic executions? Well, with no PSS, using an (t, n) secret sharing scheme, a service can tolerate up to t-1 compromised servers during the entire lifetime of the service, because any more failures could lead to the exposure of the secret. With a PSS, we know that the PSS refreshes all the shares, so that old shares become useless. Now an adversary has to gather enough shares (at least t) between two executions of the PSS, which obviously makes the attackers' job more difficult. The secret remains confidential if fewer than t servers could be compromised from the start of one PSS to the end of the next PSS.

To show how a proactive scheme can be achieved, let's study a simple example first. We first assume that an adversary can only break into a server and have access to information stored or collected by that server. The adversary cannot change the code of the server. Suppose we have a simple (2, 2) sharing scheme. To generate two shares for secret s, we randomly select s1 and s2, so that s1 + s2 = s. We want the two servers with shares s1 and s2 to change their shares to s1' and s2', so that these two shares remain an (2, 2) sharing of the same secret s and these two shares are independent from the old shares (cannot be inferred from the old shares). The proactive secret sharing can be performed in the following steps:

1. Server 1 generates two sub-shares s11 and s12 from its share s1 using the same secret sharing scheme as the one used to generate s1 and s2 from s; that is, server 1 randomly selects two sub-shares s11 and s12, so that s1 = s11 + s12,. Server 2 does the same thing to s2: It randomly generates two sub-shares s21 and s22, so that s2 = s21 + s22.
2. Server 1 sends s12 to server 2 through a certain secure channel. Server 2 sends s21 to Server 1.
3. Server 1 has both s11 and s21 and can add them up to get a new share s1' = s11 + s21. Server 2, on the other hand, has both s12 and s22 and can generate a new share s2' = s12 + s22. Now we show that s1' and s2' constitute a (2, 2) sharing. The sum of these two shares is the sum of all the four sub-shares, which is the sum of s1 and s2, which is s.

These two shares are independent from the old ones because these sub-shares are generated randomly. Also, no server knows the secret during the entire process. Server 1 generates s11 and s12 and learns s21 from server 2, but server 1 never knows s22 and thus does not know s2' or s. Server 2, on the other hand, never knows s11, and thus does not know s1' or s. (see Figure 3)

The core properties of pro-active secret sharing:

- To renew existing shares without changing the secret, so that previous exposures of shares will not damage the secret (old shares will become useless).
- To recover lost or corrupted shares without compromising the secrecy of the shares.

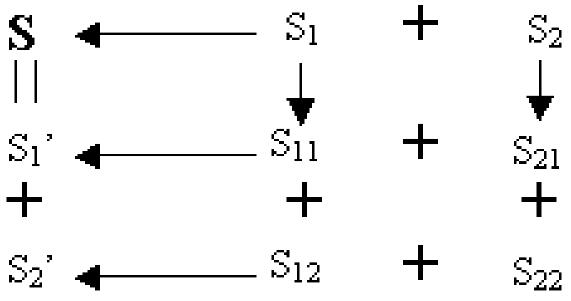


Fig 3: Methodology Used To Renew Share

This should be performed without, of course, any information-leak or any secret change.

4.1 Proactive Model Requirements

1. An adversary can reveal at most t-1 shares in any time period (where t-1<n/2. this guarantees the existence of t honest shareholders at any given time). This time period should be synchronized with the share-renewal protocol.
2. Authenticated broadcast channel.
3. Authenticated and secret communication channels between each two participants.
4. Synchronization: the servers (shareholders) can access a common global clock so that the protocol can be applied in a certain time period.
5. Shares can be erased: every honest server (shareholder) can erase its shares in a manner that no attacker can gain access to erased data.

We assume that the adversary is computationally bounded, so that it cannot break the public key encryption and the verifiable secret sharing mechanism.

4.2 Basic Share Renewal Protocol

The goal here is to renew the shares without the Dealer's involvement. (as the Dealer might not exist anymore). The shareholders should agree on a new polynomial with the same secret *s* without revealing the secret, the old polynomial or the new polynomial. At the end of this protocol, each shareholder will obtain a new share on the new *t-1* polynomial. The assumption in this protocol is that each shareholder remembers his/her old share.

We assume an initial stage where a secret *s* is encoded into *n* shares using Shamir's secret sharing scheme. Each participant holds his/her share *f(i)* for some *t-1* degree polynomial *f(x).* After the initialization, at the beginning of each time period, all honest servers/shareholders trigger an update phase in which the servers perform a share renewal protocol.

The protocol at the beginning of a time period is as follows:

1. Each i'th shareholder $i \in [1 \ldots n]$ randomly picks *t-1* numbers from the finite field. These numbers define a polynomial $P_i(X)$ of degree *t-1* whose free coefficient is zero ($P_i(0) = 0$).
2. Each i'th shareholder distributes the shares of $P_i(X\ )$ using VSS among the shareholders.
3. Each i'th shareholder receives the following shares: $P_1(i), \ldots, P_n(i)$ including his own made share $P_i(i)$) and computes his/her new share by adding his old share- *f(i)* to the sum of the new n shares. Mathematically speaking: $h(i) = f(i) + \sum_{c=1}^{n} P_c(i)$.
4. Each i'th shareholder erases his/her old share - *f(i).*

This protocol solves the share renewal problem against a passive adversary who may learn the secret information available to corrupted shareholders, but where all the shareholders follow the predetermined protocol.

4.3 Detection of Corrupted Share

In a pro-active secret sharing system, participating shareholders must be able to make sure whether shares of other shareholders have not been corrupted or lost, and restore the correct share if necessary. Otherwise, an adversary could cause the loss of the secret (by destroying *n-(t-1)* shares). The goal in this section is to present a mechanism for detection of corrupted shares.

There are obvious situations in which there is a high probability that the share is ruined, e.g. a disk crash, but how would anyone find out that a hacker penetrated his/her computer, revealed his/her share and changed it? The idea is to save some fingerprint for each share that is common to all the shareholders, so that periodically, shareholders can compare shares (using secure broadcast).

In order to implement the distributed verifiability of shares, a basic feature is added to the previous protocol. In each time period, each shareholder stores the **encryptions** of all the shares he/she received from the other shareholders. This is achieved as follows:

- Perform the non-interactive VSS, so the encryption of the initial shares will be stored at each shareholder.
- Using the homomorphic property, each i'th shareholder updates his/her set of encrypted shares by computing for every j: $E(h(i)) = E(f(i)) \times \prod_{m=1}^{n} E(P_m(j))$. Actually, this product is computed using only update shares corresponding to well behaved shareholders.

4.4 Reconstruction of Lost/ Corrupted Share

This is a fundamental phase in the proactive scheme, because without it, this scheme would not be secure against adversaries who disable some shareholders from performing the required protocol.

The basic idea is to send a shareholder - *r,* who lost his share, information that will help him recover it without the Dealer's involvement. A simple solution is to let each shareholder send his/her own share to *r*; that would allow *r* to recover the polynomial *f(X),* and then substitute *r* and recover his lost share *f(r).* However, this would expose the secret *s* to *r.*

The algorithm for reconstructing the f(r) share is as follows:

1. Each i'th shareholder $(i \in [1 \ldots r-1], [r+1, \ldots, n])$ randomly chooses a polynomial $P_i(X)$ of degree *t-1* where $P_i(r) = 0$ and $P_i(0) \neq 0$.
2. Each i'th shareholder (except for the r'th shareholder) distributes shares of $P_i(X)$: $P_i(1), \ldots, P_i(n)$ using VSS among the shareholders (except for the r'th shareholder).
3. Each i'th shareholder (except for the r'th shareholder) receives $P_i, \ldots, P_{r-1}(i), P_{r+1}(i), \ldots, P_n(i)$ and calculates his/her new share for *r*: $h(i) = f(i) + \sum_{c=1}^{r-1} P_c(i) + \sum_{k=r+1}^{n} P_k(i)$ and sends it encrypted to *r.*
4. The r'th shareholder decrypts these shares and interpolates them to recover *f(r).* He/she receives a new polynomial in which the r'th share has the same value as the old lost share: *h(r) = f(r).*

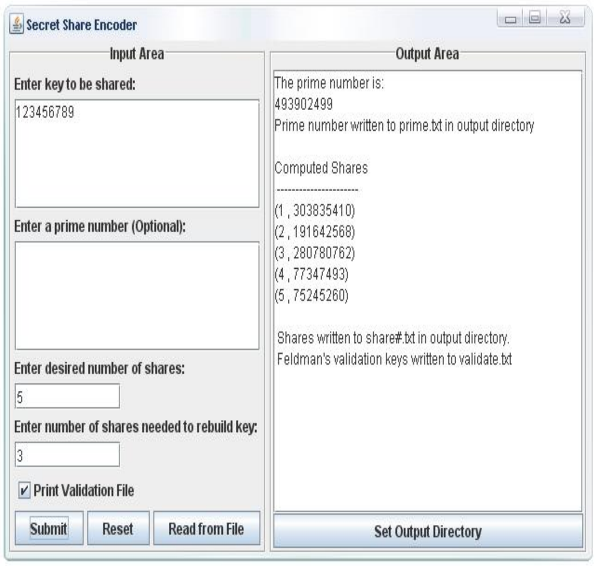
Note that this protocol is secure only against an adversary that eavesdrops on *t-1* or less shareholders but cannot change their behavior [8].

5. IMPLEMENTATION OF SHAMIR'S SECRET SHARING

The Java implementation of Shamir's scheme involves two programs. The first create a Swing form with labeled JTextFields that will accept a key, the threshold values of *n* and *t*, and the prime number *p* that will define the modulus in which the program will work. The program will then generate a polynomial function s(x) of degree *m-1* with random coefficients where the constant term is the secret key. This can be done using the simple random function for integers and storing the coefficients into an array. For BigIntegers, Java provides a constructor that will generate random values. Once the polynomial is built, the m shares (x, y) will be constructed by choosing x=1…N and y being s(x). The program will then print out all m shares and the polynomial into a JTextArea. Presumably the dealer can then take the shares and distribute them electronically or in person. The second program will be the complement to this one. It will begin by displaying JTextFields that will accept the prime number *p* along with the minimum number of shares, *t,* needed to rebuild the message. It will then display m JTextFields where each participant can enter their share. The program will then perform the necessary calculations and return the value of the key/secret.

Program Name: SecretEncoder.

Description: This program generates secret shares from an entered number based on Shamir's algorithm. The program begins by presenting the user with a GUI divided into left and right halves. On the left side is a data entry panel where the user can supply the secret key (a large number), a prime number (q) for performing the necessary modulus arithmetic, the number of shares desired, and the minimum number of shares required to recreate the secret. The entry panel also includes a button that allows the user to read the key and prime number from a file. The file should have the key on the first line and the prime # on the second. The right half of the GUI holds an output panel that displays the prime number and the shares produced. Below the output area is a button that lets the user write the shares to a directory with one share per file. This simplifies share distribution because the user can encrypt each file and send it to the appropriate party. The prime number is also written to its own file so that it can be sent to all participants. Update: The encoder has been altered so that if a checkbox is enabled on the data entry panel, the program will generate a file that enables shares to be evaluated using Feldman's method. This file contains a large prime number p such that p-1 is multiple of q, a value g, and g raised to the power of each of the Shamir polynomial coefficients. This last step is used to hide the coefficients using the know difficulty of solving the discreet log problem. The program has also been altered so that the user no longer needs to enter a prime number. If one is not supplied, the program will generate one larger than the key and write it to the output area and the file. Pressing the submit button will cause the initial GUI to be replaced by one where the shares can be entered into a set of text fields. The number of fields is dynamic and is set according to the number of shares required to rebuild the key. For convenience, each of the shares can also be read from a file by selecting the button next to each one. Once entered, one can press the clear, restart, or find key buttons. The clear button empties all the fields and the restart button restores the initial screen so that the program can be used for a different set of shares. Lastly, the find key button will compute the secret key based on the shares provided. It will always provide a key, but if any of the shares is not correct then the key will be wrong. If one chose to validate the shares, the program will check each one before computing they key and warn users of any mistakes. It will clearly indicate what share was erroneous so that the user(s) can take action.



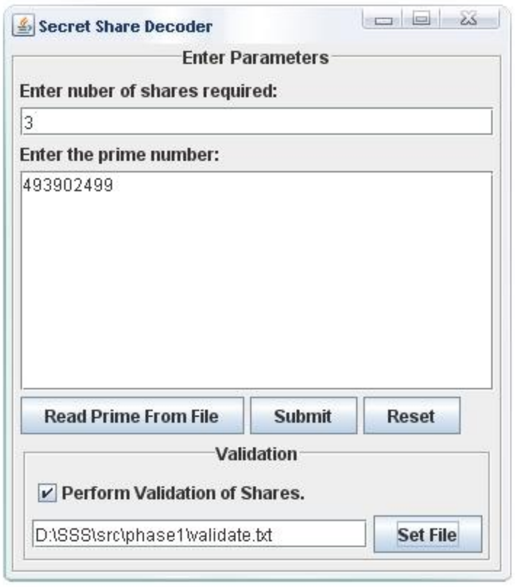
Enter key to be shared:
123456789
Enter a prime number (Optional):
Enter desired number of shares:
5
Enter number of shares needed to rebuild key:
3
✓ Print Validation File

```
The prime number is:
493902499
Prime number written to prime.txt in output directory

Computed Shares
--------------------
(1 , 303835410)
(2 , 191642568)
(3 , 280780762)
(4 , 77347493)
(5 , 75245260)

Shares written to share#.txt in output directory.
Feldman's validation keys written to validate.txt
```

Fig 4: Secret Share Encoder

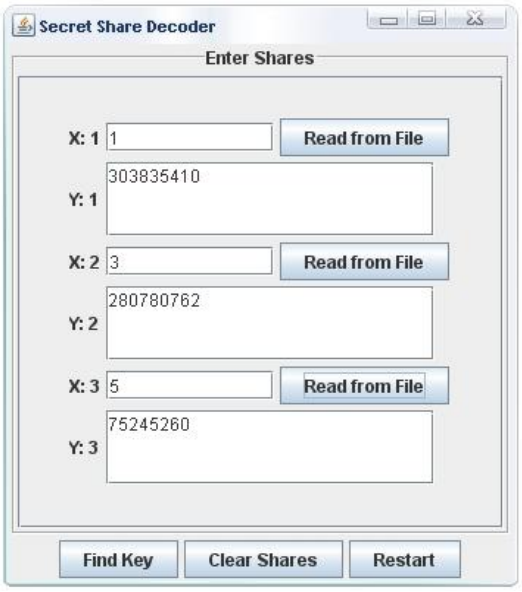
Program Name: SecretDecoder

Description: This program reads in a set of shares generated using Shamir's algorithm and uses them to compute the secret key. The program is written in a general fashion and may be used to recover the key as long as the prime number and a sufficient number of shares are known. The program makes no effort to gather the shares. They must be supplied as files or entered by the user or users. The program begins by displaying a GUI with fields to enter the prime number and the number of shares. The prime number may be read from a file by pressing the button located next to the field. The initial GUI also displays a checkbox for indicating whether or not the shares should be validated using Feldman's technique. If checked, one can supply the location of the validation file by pressing a button. Finally, one can clear the information using the reset button or click the submit button to enter the shares.



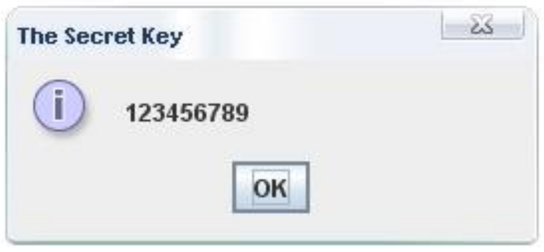
Enter nuber of shares required:
3
Enter the prime number:
493902499
✓ Perform Validation of Shares.
D:\SSS\src\phase1\validate.txt

Fig 5(a): Secret Share Decoder



X: 1 1
Y: 1 303835410
X: 2 3
Y: 2 280780762
X: 3 5
Y: 3 75245260

Fig 5(b): Secret Share Decoder



123456789

Fig 5(c) The Secret Key

6. IMPLEMENTATION OF ONLINE SECRET SHARING

The online secret sharing includes two parts. The first one is the Sever side program and the second one is the client side program.

The secret server create a Swing form with labeled JTextFields that will accept a key, the threshold value of *t*, and the prime number *p* that will define the modulus in which the program will work. The program will then generate a polynomial function s(x) of degree *m-1* with random coefficients where the constant term is the secret key. This can be done using the simple random function for integers and storing the coefficients into an array. For BigIntegers, Java provides a constructor that will generate random values. Once the polynomial is built, the m shares (x, y) will be constructed by choosing x=1…N and y being s(x). The program will then send out all m shares to the clients requesting for a share. The server is an ongoing running program. It continuously monitors the server port for client request. Whenever it gets request to generate the secret, it waits for the client to send all required number of shares. After receiving the shares it generates the secret and then sends it to the clients requested for the secret. The program has a log file option to monitor the ongoing processes.

The client side has two programs. The first one is for requesting the server to send share and the second one is for sending the share to the server. The first program takes the IP address of the server and the port number required to connect with the server. When the connection is complete then it request for its share and after receiving the share it saves the share to a text file. The second program sends share to server to reveal the secret. Like the first one it takes the IP address and the port to connect with the server. Then it sends the share previously saved by the first program to the server and wait for the server response. If the server got all the share then this program receives the secret otherwise it gets a message that “Server is waiting for other shares.” Whenever server got all the share this program can receive the secret from the server.

For the implementation of this part we have used the *java socket programming*.

6.1 Simulation of Proactive Secret Sharing

We have already mentioned the problem with the online secret sharing. Proactive secret share can resolves the problem. Here we have given the protocols needed for the proactive secret sharing. To prove the protocols we have simulate the proactive secret sharing using the Java Cryptography Architecture. Cryptix is one of the JCA providers which is open source and widely used by the Java Community. We have used Cryptix to simulate the protocols for the proactive secret sharing. The simulation shows us that all the protocols are correct. And if the protocols are implemented, it will provide the best security for any kind of data including the cryptographic keys.

7. CONCLUSIONS

This thesis has focused on a very important cryptographic primitive – secret sharing scheme. A secret sharing scheme starts with a secret and then derives from it certain shares which are distributed to some users. The secret may be recovered only by certain predetermined groups which belong to the access structure. Secret sharing schemes have appeared as an elegant solution for the problem of safeguarding cryptographic keys but their applications include now threshold cryptographic protocols and some e-voting or e-auction protocols. We have reviewed the most important secret sharing schemes for different access structures (general, threshold, online, proactive,). Some very interesting and useful extended capabilities have been also surveyed so that the applications can be easily comprehensible.

Our major contribution consists in the application of the general variant of the Lagrange Polynomial theorem in designing several classes of secret sharing. We consider that the proposed secret sharing schemes provide the flexibility for performing a required compromise between the size of the shares and the level of security.

We have pointed out that the secret sharing schemes based on the general variant of the Lagrange polynomial theorem have some interesting and useful features as multiplicative and homomorphic properties which make them suitable for threshold cryptography.

Java Cryptography Architecture (JCA) is a successful object-oriented framework for conventional public key cryptography and has been widely used. In this thesis, we extend the JCA framework to integrate the threshold cryptography, a branch of group-oriented public key cryptography which has been shown to be a very useful tool to improve system security. Under such a framework extension, an application can easily change its threshold cryptography providers at run time without changing its source codes. Since our extension follows the JCA design principle, use threshold cryptography. An example provider is implemented to show the feasibility of the framework extension. It is our belief that this practice will help speed up the adoption of threshold cryptography.

8. REFERENCES

[1] Fredrik Olsson, “A Lab System for Secret Sharing”, 2004.
[2] N. Ferguson & B. Schneier. Practical Cryptography. 2003, pp. 358-360.
[3] R. Anderson, R. Needham & A. Shamir. “The Steganographic File System." in D. Aucsmith (ed.) Information Hiding. Second International Workshop. 1998, pp. 73-82.
[4] Lidong Zhou. Secret Sharing. Secret Sharing. Retrieved August 01, 2013, from http://www.cs.cornell.edu/courses/cs513/2000sp/SecretSharing.html
[5] Secret Sharing Scheme. Retrived Auguest 02, 2013, from http://www.jetico.com/web_help/bc8/html/08_1_new_key_generator/03_SSS.htm
[6] A. Shamir, How to share a secret, Communications of the ACM 22 (1979), 612-613.
[7] What is "Secret Sharing"? Retrived Auguest 02, 2013, from http://point-at-infinity.org/ssss/
[8] V. Nikov, S. Nikova. On Proactive Secret Sharing Schemes, SAC’04, LNCS 3357, 2004, pp. 314-331.