AN UNCHANGED SHADOW-BASED SECRET SHARING SCHEME

By

NUR HADISUKMANA

Sarjana Fisika

University of Indonesia

Jakarta, Indonesia

1988

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 1995

AN UNCHANGED SHADOW-BASED

SECRET SHARING SCHEME

Thesis Approved:

[illegible]
Thesis Adviser

Blayne E. Mayfield

H. Lu

Thomas C. Collins
Dean of The Graduate College

PREFACE

The purpose of this study was to propose a different construction for a secret sharing scheme that maintains recycleable pieces of information, which are called shadows, about a secret information item. In this study, the properties of the existing established secret sharing schemes were analyzed and a solution for the common weakness found in these schemes was proposed. A network program was also designed and implemented as an example of the proposed scheme.

The constructions that have been introduced to improve the secret sharing scheme's performance generally suffer from the common weakness of incurring the overhead of regenerating and redistributing different shadows when a new secret key (information) is created. The proposed scheme developed in this study was intended to resolve the above-mentioned weakness by preserving the original shadows, such that they could be used repeatedly independently of the creation of the secret key (information). This feature was called shadow recycleability. The proposed scheme constructed a certain polynomial for determining the secret, $Y(x) = a_1 x + a_2 x^2 + \ldots + a_r x^r$, where x refers to a shadow and x, x^2, ..., x^r are computed using modulo a prime number m. Besides having the shadow recycleability property, the proposed scheme was also intended to serve as a general access structure secret sharing scheme by dividing m into several pieces of information I_ms, and by distributing them, together with the coefficients of Y(x), to participants, who are

grouped into two unique subsets U_1 and U_2 such that only the qualified subsets of the participats can reveal the prime number m and reconstruct the polynomial Y(x). The proposed scheme was implemented in a network program called Secret Conference, which allows a certain number of users to converse secretly.

The proposed scheme, however, has a restriction. The size of the first unique subset U_1 is limited to two members. The restriction exists because of the method used to distribute the coefficients of Y(x). This method requires (as done on the prime number m) that only the qualified subsets of the participants can reconstruct the polynomial Y(x). If the size of U_1 is increased to be three or more, the distribution of the coefficients of Y(x) (using the method described in this report) will result in a condition where there is at least one unqualified subset of the participants that can reconstruct the polynomial Y(x). This situation makes the unqualified subset (in guessing the secret key K) focus only on finding the correct value of one type of information left unknown: the prime number m, although guessing the correct value of m is difficult.

ACKNOWLEGMENTS

All praises be to God.

I would like to express my sincere appreciation and thanks to my major adviser Dr. Mansur H. Samadzadeh for his invaluable and consistent support and guidance from the early stages of this work. My sincere thanks are also due to Drs. Blayne E. Mayfield and Huizhu Lu for providing their precious time as my graduate commitee members.

I would like to thank the Goverment of Indonesia for providing funds, through the Agency for the Assessment and Application of Technology, for my graduate studies. I would also like to express my thanks to the National Atomic Energy Agency of Indonesia for granting me the permission to pursue higher education.

Finally, I express my sincere thanks to my family, my wife Rina Dwi Septika and my daughter Khadijah, for their love, support, encouragement, and patience during my graduate studies. Special thanks go to my parents-in-law for their support and encouragement at times of difficulty.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Since being independetly introduced by Blakley [Blakley79] and Shamir [Shamir79] about 15 years ago, the secret sharing scheme, originally called a threshold scheme, has been implemented by many researchers. The basic idea of the scheme is to maintain a piece of secret information by dividing it into several parts, called shadows, and distributing the shadows to a set of n participants in such a way that a certain subset of the participants can recover the secret information by pooling the shadows they have [Simmons92]. The collection of subsets of the participants that can recover the secret information is called an access structure [Jackson94]. If the access structure contains all subsets whose size is at least t, then the scheme is called a (t,n) threshold scheme [Jackson94] [Stinson93].

A number of (t,n) threshold schemes have been developed implementing the notion that, by pooling any t or more shadows, the secret information itself can be revealed, while gaining t - 1 or fewer shadows will not be able to recover it [Simmons89]. Other researchers have developed a general access structure for secret sharing schemes. The idea of general access structure is to limit the subsets of the participants in recovering the secret information such that only certain specified subsets of the participants can recover it, while the unspecified subsets cannot [Blundo93]. These schemes, however, have a weakness that whenever the secret information is revealed, all of its shadows become worthless [Harn93]. As a consequence, new shadows should be created and distributed if a new piece of secret information is generated.

The main objective of this thesis was to introduce a different approach (scheme) that can resolve the above-mentioned drawback by preserving all of the shadows already given to the participants, so that new shadows need not be distributed to the participants each time a new secret information is created. The proposed scheme is also designed to support the general access structure secret sharing scheme.

The rest of this thesis is organized as follows. Chapter II provides a review on literature related to the proposed scheme. Chapter III discusses the work of the proposed scheme. Chapter IV gives an analysis of the proposed scheme and its comparison with Shamir's scheme. An implementation of the proposed scheme in a network program, which is called Secret Conference, is discussed in Chapter V. Finally, Chapter VI outlines the summary and future work of the proposed scheme.

CHAPTER II

LITERATURE REVIEW

2.1 Linear Algebra

This section gives a brief review of the some concepts from linear algebra. Such concepts are used in obtaining the coefficient values of a polynomial Y(x), which is described in Section 3.1. An equation in n variables x_1, x_2, ..., x_n, is said to be a linear equation if all of its variables appear in their first power forms [Anton81].

$$a_1 x_1 + a_2 x_2 + \dots + a_n x_n = b \tag{2.1}$$

To obtain the values of the n variables, there should be n linear equations. The set of n linear equations is called a system of n linear equations or a linear system [Anton81].

$$\begin{aligned} a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n &= b_1 \\ a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n &= b_2 \\ &\ \vdots \\ a_{n1} x_1 + a_{n2} x_2 + \dots + a_{nn} x_n &= b_n \end{aligned} \tag{2.2}$$

Such a linear system can be expressed as a matrix equation

$$\begin{bmatrix} a_{11} & . & . & . & a_{1n} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ a_{n1} & . & . & . & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ . \\ . \\ . \\ b_n \end{bmatrix} \tag{2.3}$$

or [**A**] [**X**] = [**B**].

If the values of the elements of matrices [**A**] and [**B**] are known, the values of matrix [**X**] can be determined.

$$\begin{aligned} [\mathbf{A}]\,[\mathbf{X}] &= [\mathbf{B}] \\ [\mathbf{A}]^{-1}\,[\mathbf{A}]\,[\mathbf{X}] &= [\mathbf{A}]^{-1}\,[\mathbf{B}] \\ [\mathbf{I}]\,[\mathbf{X}] &= [\mathbf{A}]^{-1}\,[\mathbf{B}] \\ [\mathbf{X}] &= [\mathbf{A}]^{-1}\,[\mathbf{B}] \end{aligned} \tag{2.4}$$

where [**I**] is identity matrix and $[\mathbf{A}]^{-1}$ is the inverse of matrix [**A**].

There are two methods for obtaining the inverse matrix $[\mathbf{A}]^{-1}$. The first method determines the inverse matrix $[\mathbf{A}]^{-1}$ based on the following equation.

$$[\mathbf{A}]^{-1} = \frac{1}{|\mathbf{A}|}\ \mathrm{Adj}(\mathbf{A}) \tag{2.5}$$

where |**A**| is the determinant of matrix [**A**] and Adj(**A**) is the adjoint matrix of [**A**] [Hershey86].

The second method obtains the inverse matrix $[\mathbf{A}]^{-1}$ using a sequence of row operations on matrix [**A**] which changes [**A**] into [**I**] and changes [**I**] into $[\mathbf{A}]^{-1}$ in a rectangular matrix [**C**] [Hershey86] [Anton81].

$$[\mathbf{C}] = [\mathbf{A} \mid \mathbf{I}] \tag{2.6}$$

To find the inverse matrix $[\mathbf{A}]^{-1}$, a sequence of row operations can be performed as follows. Suppose that there is an integer i, where i is initialized to 1.

1. Determine the value of element C_{ii}. If $C_{ii} = 0$, exchange row i with another row whose value in its ith column is not zero.

2. Divide all elements in row i by the value of C_{ii}.

3. Multiply all elements in row i by the negative of C_{ji} and add the resulting values to the corresponding elements in row j so that the value of C_{ji} will be zero. Repeat these operations until all elements in column i are zero except for C_{ii}.

4. Repeat all steps, starting from Step 1, for i = 2, 3, ..., n.

Suppose, as a simple example, that the entries of matrix **[A]** are as given below.

$$[\mathbf{A}] = \begin{bmatrix} 2 & 4 \\ 3 & 7 \end{bmatrix}$$

To find the inverse matrix of **[A]** using the above algorithm, matrix **[C]** is constructed.

$$[\mathbf{C}] = \left[\begin{array}{cc|cc} 2 & 4 & 1 & 0 \\ 3 & 7 & 0 & 1 \end{array}\right]$$

The sequence of row operations for the matrix **[C]** is as follows (assuming that initially i is 1).

- In Step 1, since $C_{ii} = C_{11} = 2$ ($C_{11} \neq 0$), proceed to Step 2.
- In Step 2, all entries in row i = 1 are divided by $C_{ii} = C_{11} = 2$. The result is shown below.

$$[\mathbf{C}] = \left[\begin{array}{cc|cc} 1 & 2 & \frac{1}{2} & 0 \\ 3 & 7 & 0 & 1 \end{array}\right]$$

- After applying the operations described in Step 3, where j = 2 and $C_{ji} = C_{21} = 3$, the result is as follows.

$$[\mathbf{C}] = \left[\begin{array}{cc|cc} 1 & 2 & \frac{1}{2} & 0 \\ 0 & 1 & -\frac{3}{2} & 1 \end{array}\right]$$

- After repeating all steps starting from Step 1 for i = 2, where j = 1, $C_{ji} = C_{12} = 2$, the final result is shown below.

$$[\mathbf{C}] = \left[\begin{array}{cc|cc} 1 & 0 & \frac{7}{2} & -2 \\ 0 & 1 & -\frac{3}{2} & 1 \end{array}\right]$$

where the second part of matrix **[C]** is the inverse matrix of **[A]**.

Compared to the first method, the second one is simpler and easier to implement, particularly when the size of the matrix is large (i.e., 10 or more). Therefore, the second method is used for matrix operations performed in the proposed scheme.

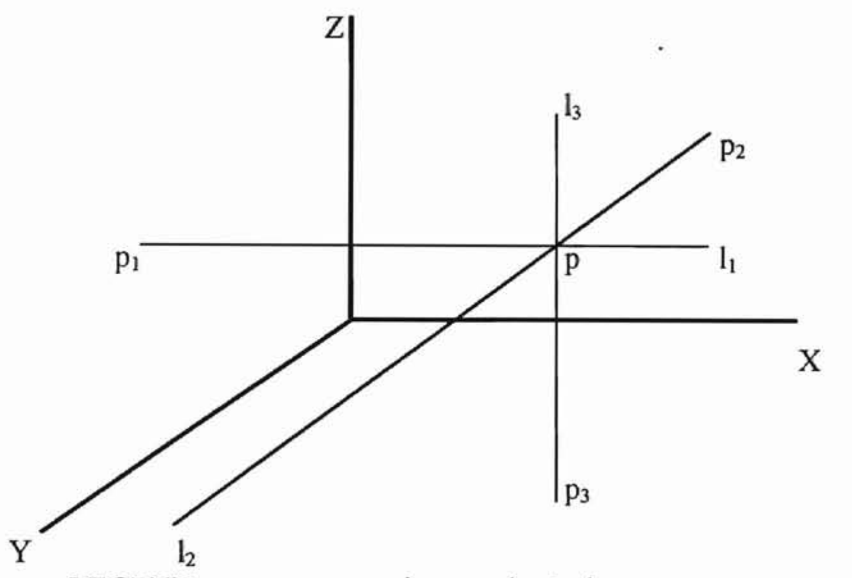
2.2 Secret Sharing Scheme

Secret sharing scheme is a method to give a set of n participants pieces of information, called shadows, of a shared secret key K in such a way that t out of the n participants, $t \leq n$, can recover K by pooling the shadows that they have [Simmons92]. This scheme is also called a (t,n) threshold scheme. The first (t,n) threshold scheme was introduced in 1979 by Shamir [Shamir79] and Blakley [Blakley79].

2.2.1 Blakley's Threshold Scheme

Blakley used a projective geometric model for his scheme. Simmons gave a simple and clear example illustrating Blakley's (2,3) threshold scheme [Simmons92]. In the example, a secret key K is viewed as a point p located in three dimesional space, $p = (x_p,y_p,z_p)$. Shadows of K, which are given to a set of 3 participants, are shadows of p on YZ, XZ, and XY planes, say, $p_1 = (0,y_p,z_p)$, $p_2 = (x_p,0,z_p)$, and $p_3 = (x_p,y_p,0)$, respectively. Since p_i, i = 1, 2, or 3, is a projection point of p, a vertical line l_i can be drawn from p_i so that the line goes through p. Two of the three participants can determine p by combining their lines since the intersection point of the combined lines is p, as shown in Figure 1.

Blakley's construction of the scheme is actually not in a Euclidean space E^n but in a finite projective space PG(v,p), where v indicates the dimension of the projective space and



LEGEND: p_1, p_2, p_3 points on the 3 planes
l_1, l_2, l_3 lines parallel to the axes

Figure 1. Blakley's (2,3) threshold scheme model
(Source: [Simmons92])

p is a prime number. The shadows can thus be viewed as a projection of the secret key (information) from a point in one plane onto another plane. This construction will result in v arbitrarily chosen hyperplanes.

2.2.2 Shamir's Threshold Scheme

Shamir, on the other hand, introduced his (t,n) threshold scheme based on polynomial interpolation [Shamir79]. The scheme chooses a polynomial Y(x) of degree t - 1 over a finite field GF(p), where t is the smallest number of participants that can recover the secret key K, and p is a prime number. It also selects a secret key K and assigns it as the first coefficient of Y(x).

$$Y(x) = (a_0 + a_1 x + \dots + a_{t-1} x^{t-1}) \bmod p \qquad (2.7)$$

where $a_0 = K$, p is a prime number larger than K, and the coefficients a_1, a_2, ..., and a_{t-1} are arbitrarily chosen. K is divided into K_1, K_2, ..., K_n by evaluating Y(x) at n distinct values of

$x_1, x_2, \ldots,$ and x_n, where n is the total number of participants.

$$K_i = Y(x_i) \quad \text{for } i = 1, 2, \ldots, n$$

In Shamir's scheme, a shadow is defined as a pair of values (x_i, K_i). Since the total number of participants in the scheme is n, n shadows should be created. Each shadow, together with the prime number p, is given to every participant.

By pooling t out of n shadows, $(x_1, K_1), (x_2, K_2), \ldots, (x_t, K_t)$, Y(x) can be reconstructed using the Lagrange interpolation equation [Denning82].

$$Y(x) = \left(\sum_{i=1}^{t} K_i \prod_{j=1, j \neq i}^{t} {}^{(x - x_j)} \!\big/ {}_{(x_i - x_j)} \right) \bmod p \qquad (2.8)$$

where (x_i, K_i) is one of the t shadows, x_j is x value of the other t-1 shadows, and $x_j \neq x_i$. Consequently, K can be recovered by evaluating Y(0) since $Y(0) = a_0 = K$.

Example:

Suppose that there is an integer value K that is kept secret. K is shared among 5 participants in such a way that 3 out of the 5 participants can recover K by gathering the shadows of K that they have. The five shadows are determined by using Shamir's (t,n) threshold scheme (Source: [Denning82, p. 181]).

Let $t = 3$, $n = 5$, $p = 23$ and $K = a_0 = 21$. We choose at random $a_1 = 5$ and $a_2 = 11$, so that

$$Y(x) = (21 + 5x + 11x^2) \bmod 23$$

If $x = 1, 2, \ldots,$ and 5 are chosen, the shadows will be.

$$\begin{aligned}
K_1 &= Y(1) = (21 + 5 + 11) \bmod 23 = 14 \\
K_2 &= Y(2) = (21 + 10 + 44) \bmod 23 = 6 \\
K_3 &= Y(3) = (21 + 15 + 99) \bmod 23 = 20 \\
K_4 &= Y(4) = (21 + 20 + 176) \bmod 23 = 10
\end{aligned}$$

$$K_5 = Y(5) = (21 + 25 + 275) \bmod 23 = 22$$

We can reconstruct Y(x) by choosing 3 out of the 5 shadows, say K_1, K_2, and K_4.

$$\begin{aligned}
Y(x) &= [14 * \frac{(x-2)(x-4)}{(1-2)(1-4)} + 6 * \frac{(x-1)(x-4)}{(2-1)(2-4)} + 10 * \frac{(x-1)(x-2)}{(4-1)(4-2)}] \bmod 23 \\
&= [14 * \mathit{Inv}(3,23) * (x^2 - 6x + 8) + 6 * \mathit{Inv}(-2,23) * (x^2 - 5x + 4) + 10 * \\
&\quad \mathit{Inv}(6,23) * (x^2 - 3x + 2)] \bmod 23 \\
&= [14 * 8 * (x^2 - 6x + 8) + 6 * 11 * (x^2 - 5x + 4) + 10 * 4 * (x^2 - 3x + 2)] \\
&\quad \bmod 23 \\
&= [20 * (x^2 - 6x + 8) + 20 * (x^2 - 5x + 4) + 17 * (x^2 - 3x + 2)] \bmod 23 \\
&= [57x^2 - 271x + 274] \bmod 23 \\
&= 11x^2 + 5x + 21 \\
Y(0) &= a_0 = K = 21
\end{aligned}$$

Note: Inv (a,b) is an extended Euclid's algorithm to compute the inverses of a modulo b.

The extended algortihm is shown below [Source: Denning82, p. 44]

Algorithm *Inv* (a,n)

```
begin {Return x such that ax mod n = 1}
        $g_0 := n;\ g_1 := a;$
        $u_0 := 1;\ v_0 := 0;$
        $u_1 := 0;\ v_1 := 1;$
        $i := 1;$
        while $g_i \neq 0$ do {$g_i = u_i n + v_i a$}
              begin
                    $y := g_{i-1}$ div $g_i$;
                    $g_{i+1} := g_{i-1} - y * g_i$;
                    $u_{i+1} := u_{i-1} - y * u_i$;
                    $v_{i+1} := v_{i-1} - y * v_i$;
                    $i := i + 1$;
              end;
              $x := v_{i-1}$;
              if $x > 0$ then inv := x else inv := x + n;
end
```

It should be noted that although Blakley and Shamir used different constructions, their schemes give similar properties [Simmons92] as listed below.

1. t out of n shadows will suffice to recover the secret key (information) K.

2. t - 1 or fewer shadows can never reveal K; therefore, although up to t - 1 of the participants are unavailable, the secret key K still can be recovered.

2.2.3 General Access Structure Secret Sharing Scheme

Since Blakley and Shamir proposed their schemes, a number of researchers have given more attention to the secret sharing scheme. They have developed the scheme in the form of a general access structure. A general access structure secret sharing scheme is defined as a way to divide and distribute the secret key K to a set of n participants such that only the qualified subsets of the participants can recover K, while those who are unqualified cannot recover it [Blundo93]. The collection of the qualified subsets is called an access structure, denote by Γ [Jackson94]. If the access structure contains all subsets whose size is at least t, then the scheme is called a (t,n) threshold scheme [Jackson94] [Stinson93]. It is clear therefore that a general access structure secret sharing scheme (general access structure scheme for short) is a generalization of the (t,n) threshold schemes.

A general access structure scheme is said to be perfect if the following properties are satisfied [Blundo93] [Stinson93].

1. If a qualified subset of the participants gather their shadows, they can recover the secret key (information) K.
2. If an unqualified subset of the participants gather their shadows, they obtain no information about K.

A general access structure scheme is monotone if there is a subset D of the participants that is a superset of a qualified subset B of the participants, then D is also a qualified subset that can recover K [Stinson93], which can be expressed by an equation below.

$$\text{if } B \in \Gamma \text{ and } B \subseteq D \subseteq P, \text{ then } D \in \Gamma \qquad (2.9)$$

where P is a set of all participants and Γ is an access structure or the set of qualified subsets. In a general access structure scheme, there is a special and trusted participant, called dealer, who is responsible for recreating the secret key [Stinson93].

2.3 Drawback of Secret Sharing Scheme

Secret sharing scheme is useful in information security [Simmons92], in multi-user network cryptography system [Jackson94], in opening vault-lock or safety deposit box, or even any controlled action that requires the presence of several persons in order to initiate the action [Blundo93].

The scheme, however, has a drawback. Once the secret key K is recovered, all shadows of K, including those which do not involve in the recovery process, become useless [Harn93]. Consequently, whenever a new secret key K' is created, all participants should be given the corresponding shadows. To resolve the drawback, this thesis proposes a new approach in secret sharing scheme by maintaining the original shadows that have already been given to all the participants, such that the shadows can still be used to reveal the new secret key K'. This characteristic is called recycleable shadows. Another characteristic of the proposed scheme is that it supports a general access structure.

The next two chapters discuss the proposed scheme's work and other issues related to it. Chapter III discusses the construction and the work of the proposed scheme including the assumptions on which the proposed scheme is based. Chapter IV discusses an analysis of the proposed scheme and its comparison with Shamir's scheme.

CHAPTER III

PROPOSED SCHEME

This chapter discusses various aspects of the proposed scheme. First, it discusses the construction of the proposed scheme. Second, it describes the operation of the proposed scheme, which can be divided into three phases, i.e., creation, distribution, and recovery. Third, it gives a practical example implementing the proposed scheme.

3.1 Construction of the Proposed Scheme

The proposed scheme is designed using a polynomial Y(x) of degree r.

$$Y(x) = a_1 x + a_2 x^2 + \ldots + a_r x^r \tag{3.1}$$

where $r = (|U_1| + 1) * |U_2|$, and U_1 and U_2 are unique subsets described below. This polynomial is used as a template to determine the coefficient values $a_1, a_2, \ldots, a_r$ of the polynomial in the creation phase, and to obtain the secret key (information) in the recovery phase, as described in Section 3.2. The value $r = (|U_1| + 1) * |U_2|$ is chosen in order to minimize the size of the information distributed to the participants and to guarantee that every participant, in receiving the distributed information, misses at least two coefficients of Y(x), so that the difficulty in guessing the secret key (based on each participant's information) is increased.

The proposed scheme is constructed based on the following assumptions and notations.

1. Let K denote the secret key (information) to be shared, S a set of shadows of K, m a prime number (m is used instead of p, which is a prime number, to differentiate this notation from the one used in Blakley's and Shamir's schemes), I_m a piece of information relating to m, and C a set of some of the coefficients of a certain polynomial Y(x). The idea behind breaking m into I_ms and that of dividing all coefficients of Y(x) into Cs, is to protect m and Y(x) such that only members of the qualified subsets of the participants can reveal m and reconstruct Y(x). Y(x) and m are needed to derive the secret key K.

2. Let P be a set of n trusted participants, $P = \{P_1, P_2, ..., P_n\}$. The participants in P are classified into two groups, where the size of one group is smaller than that of the other. The proposed scheme sets P into two groups (U_1 and U_2) to implement the idea introduced by Simmons [Simmons92] that one member of one group U_1 can take an action on behalf of one or two members of the other group U_2. The proposed scheme defines such a capability as the *weight* of the members of U_1 over the members of U_2, and denotes it as W.

3. Let Δ be a special and trusted participant called a *dealer*, where $\Delta \notin P$ (see Subsection 2.2.3). In the proposed scheme, the dealer Δ does several things. First, it selects K, S, and m as described in Item 1 above. Second, it breaks m into pieces of information called I_ms. I_m is expressed in the following form.

$$I_m = (x, Z(x)) \qquad (3.2)$$

   where x is an arbitrary integer value and Z(x) is a random polynomial whose first coefficient equals the prime number m. Third, it sets a polynomial Y(x) (Equation (3.1)) such that Y(x) always gives the same value as K each time Y(x) is evaluated for the values of S. Finally, it distributes the specified information (i.e., some values of S, I_ms, and the coefficients of Y(x) grouped into Cs) to P through a secure channel, such that only the qualified subsets of P can recover K using the distributed information.

4. Let Γ be a monotone access structure consisting of all qualified subsets of P that can recover K (see Subsection 2 2.3 for the definition). Two of the subsets are unique subsets U_1 and U_2. The two subsets are the two groups mentioned in Item 2 above. A unique subset is defined as a subset whose members are not the members of the other unique subset, $U_1 \cap U_2 = \varnothing$. For instance, consider an access structure Γ of 5 participants A, B, C, D, and E containing the following subsets.

$$\Gamma = \{\{A,B\},\{C,D,E\},\{A,C,D\},\{A,D,E\},\{B,C,E\}\}$$

Here, {A,B} and {C,D,E} are the unique subsets U_1 and U_2 since $\{A,B\} \cap \{C,D,E\} = \varnothing$, while others are non-unique subsets since their members are members of unique subsets U_1 and U_2. The smallest size in Γ is assumed to be t, $t = |U_1|$ and $t \geq 2$, and the size of the biggest subset in Γ is assumed to be n - t, $n - t = |U_2|$ and $|U_2| > |U_1|$, where n is the total number of participants in P.

5. Since U_1 and U_2 are the smallest and the largest subsets in Γ and the members of the non-unique subsets are from U_1 and U_2, the sizes of the non-unique subsets span the range from $|U_1|$ to $|U_2|$.

$$|U_1| \leq |\alpha| \leq |U_2| \tag{3.3}$$

where $|U_1| = t$, $|U_2| = n - t$, n is the total number of participants ($n = |P|$), α is any non-unique subset, and $|\alpha|$ is the size of α. To simplify the case of setting the non-unique subsets whose sizes satisfy Equation (3.3), the proposed scheme assumes that the two following conditions hold.

A. If $(|U_1| - 1) * W + 1 \leq |U_2|$, where W is the weight of the members of U_1 over the members of U_2, then at most $(|U_1| - 1)$ members of a non-unique subset are from U_1. Otherwise, only one member of the non-unique subset is from U_1.

B. The size of a non-unique subset should also satisfy the following Equation.

$$i * W + (|\alpha| - i) \geq |U_2| \tag{3.4}$$

where α is a non-unique subset, i is number of members of α that are from U_1, W is the weight of the members of U_1 over the members of U_2, and $|\alpha|$ is the size of α.

The proposed scheme, which implements the notions described in Items (1) to (5) above, is applicable in the real world. Suppose that there is a group of 6 authorized persons in a company. Two of them are vice presidents and the rest are managers. Hierarchically, a vice president is different from a manager. Therefore, the six persons are grouped into two levels, vice presidents and managers. They have different capabilities and authorities in initiating a specified controlled action. For instance, to recover a secret combination for opening a locked vault, requires the presence of several persons. This action can be carried out by the two vice presidents, by the four managers or by several persons who are from both levels, under the condition that a person from the first level (the vice president level) is capable of taking the act on behalf of one or two persons from the second level (the manager level). If one of the two vice presidents can act on behalf of one person from the managers level, the size of the non-unique subsets is 5. And if one of the two vice presidents can act on behalf of two persons from the managers level, then the minimum size of the non-unique subsets is 4 and the maximum is 5.

3.2 Phases of the Proposed Scheme

The proposed scheme's work can be divided into three phases: the creation phase, the distribution phase, and the recovery phase. The first two phases are performed in the dealer's site, while the last phase is applied in the participants' site.

3.2.1 Creation Phase

In this phase, the dealer provides all of the information required by subsets of P in Γ to recover K (carried out in the recovery phase). The required information consists of the coefficient values of Y(x) shown in Equation (3.1), shadows of K in S, n of which are given to P (n = $|P|$), and the prime number m, which is divided into I_ms (see Section 3.1 for definitions). Therefore, the dealer first chooses a secret K and a set of r distinct integer values for S that are used as shadows of K, $S = \{ s_1, s_2, \ldots, s_r \}$, where $r = (|U_1| + 1) * |U_2|$. The dealer also selects a prime number m and a polynomial Y(x) of degree r, as described in Equation (3.1). The coefficients of Y(x) should be unique so that Y(x) will give the same value as K, when it is evaluated for the values of S.

$$\begin{array}{l}
a_1[(s_1) \bmod m] + a_2[(s_1^2) \bmod m] + \ldots + a_r[(s_1^r) \bmod m] = K \\
a_1[(s_2) \bmod m] + a_2[(s_2^2) \bmod m] + \ldots + a_r[(s_2^r) \bmod m] = K \\
\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
a_1[(s_r) \bmod m] + a_2[(s_r^2) \bmod m] + \ldots + a_r[(s_r^r) \bmod m] = K
\end{array} \tag{3.5}$$

The above equations can be expressed in matrix form.

$$[\mathbf{S}]\,[\mathbf{A}] = [\mathbf{K}] \tag{3.6}$$

where $[\mathbf{S}]$ = the r x r shadow matrix, $[\mathbf{A}] = [a_1, a_2, \ldots, a_r]^T$, and $[\mathbf{K}] = [K, K, \ldots, K]^T$. Matrix $[\mathbf{A}]$ can be determined by first finding the inverse matrix of $[\mathbf{S}]$, $[\mathbf{S}]^{-1}$, using the second method described in Section 2.1.

$$\begin{aligned}
[\mathbf{S}]^{-1}\,[\mathbf{S}]\;[\mathbf{A}] &= [\mathbf{S}]^{-1}[\mathbf{K}] \\
[\mathbf{I}]\;[\mathbf{A}] &= [\mathbf{S}]^{-1}[\mathbf{K}] \\
[\mathbf{A}] &= [\mathbf{S}]^{-1}[\mathbf{K}]
\end{aligned} \tag{3.7}$$

Having obtained values for the elements of matrix $[\mathbf{A}]$, the dealer can then obtain the coefficient values of Y(x), since the elements of $[A]$ are indeed the coefficient of Y(x), i.e., $a_1, a_2, \ldots, a_r$. These coefficients are not just useful for deriving K but they also give the description of Y(x).

The dealer now has all of the information required by Γ to recover K, and therefore the creation phase is completed.

3.2.2 Distribution Phase

Since all the information needed to recover K is available, the dealer can basically distribute the items to the members of P. However, in order to enforce the notion (as mentioned in Item (1) in Section 3.1) that m can only be revealed if members of the qualified subsets of P (in Γ) pool their knowledge of m, the prime number m that was used in deriving K should be partitioned into pieces of information called I_ms using Equation (3.2). These I_ms are then distributed to the members of P.

In partitioning m into I_ms, the dealer considers two things. First, since I_m is obtained using Equation (3.2) and the qualified subsets consist of the two unique subsets U_1 and U_2 and the non-unique subsets, the dealer must determine the number of Z(x) polynomials that are used for creating I_ms. Second, the dealer must also determine how many units of I_m should be given to each member of P. These two things should be taken care of such that the notion stated above is satisfied. It should be noted that unless stated explicitly otherwise, the words 'units of I_m' used here and later means units of distinct values of I_m.

Since the largest subset in Γ (recall the notation in Section 3.1) is U_2 and the sizes of the non-unique subsets satisfy Equations (3.3) and (3.4), the size of U_2, $|U_2| = n - t$, is the upper bound for the qualified subsets in Γ. Hence, the dealer sets a polynomial Z(x) whose degree is $|U_2| - 1 = n - t - 1$.

$$Z(x) = z_0 + z_1 x + \ldots + z_{n-t-1} x^{n-t-1} \qquad (3.8)$$

where z_0 equals the prime number m ($z_0 = m$) and the other z_is are arbitrary integer values. This polynomial is used by the dealer to divide m into I_ms, and by the unique subset U_2 and the non-unique subsets to reveal m later in the recovery phase. Equation (3.8) implicitly indicates that the number of units I_m, needed to reaveal m using the equation, must be at least the same as the total number of coefficients of the equation (as stated by Shamir [Shamir79]). Since the number of coefficients of Equation (3.8) is the same as the size of

U_2, the dealer then sets one unit of I_m for each member of U_2. Equation (3.8) above always appears in partitioning m into I_ms as described below.

For every member of U_1, whether or not the dealer sets another polynomial Z(x) and how many units of I_m should be given, depends upon the weight W of members of U_1 over members of U_2. Based on the relationship among the sizes of U_1 and U_2, and the weight W, there are three cases that can occur in partitioning m into I_ms. It should be noted that in partitioning m into I_ms (discussed below) the proposed scheme implements Shamir's model [Shamir79] (described in Subsection 2.2.2) with a slight modification on the way to reveal the prime number m as described later in the recovery phase (Subsection 3.2.3).

Case I: If the total weight of all members of U_1 is less than the number of members of U_2, $|U_1| * W < |U_2|$ (the notation is described in Section 3.1), then the dealer will set two random polynomials that are used for partitioning m into I_ms. The degrees of these polynomials are $|U_1| - 1 = t - 1$ and $|U_2| - 1 = n - t - 1$, where $|U_1| = t$ and $|U_2| = n - t$, as stated in Item (4) in Section 3.1.

$$Z_1(x) = b_0 + b_1 x + \dots + b_{t-1} x^{t-1}, \quad \text{for } U_1 \qquad (3.9)$$

$$Z_2(x) = c_0 + c_1 x + \dots + c_{n-t-1} x^{n-t-1}, \text{ for } U_2 \text{ and other subsets} \quad (3.10)$$

where $b_0 = c_0 = m$, and the other b_is and c_is are arbitrary interger values. The two polynomials above and the other Z(x) polynomials used in the next two cases are similar to the polynomial that was used in Shamir's Scheme to produce shadows of a secret key K as described in Subsection 2.2.2. Equation (3.9) is set for the unique subset U_1, while Equation (3.10) is set for the unique subset U_2 and the non-unique subsets in Γ whose members are from U_1 and U_2. The reason why the dealer sets two Z(x) polynomials is to satisfy the notion stated at the beginning of this section (the distribution phase). If the dealer sets only one Z(x) polynomial shown either in Equation (3.9) or (3.10), then the result will violate the stated notion. Suppose that the dealer sets only one Z(x) polynomial shown in Equation (3.10), then there must be at least $|U_2| = (n - t)$ units of I_m in order to reveal the prime number m. With this requirement, the unique subset U_1 will never reveal the prime number m because the maximum number of units of I_m that can be possessed by U_1 is ($|U_1| * W$) which is less than the size of U_2, $|U_2| = n - t$. This means that one subset (i.e., U_1) in Γ cannot reveal m, and this contradicts with the definition of Γ (recall the definition in Item (4) in Section 3.1 and in Subsection 2.2.3). Suppose that, on the other hand, the dealer sets only one Z(x) polynomial shown in Equation (3.9), then the minimum number of units of I_m needed for revealing m is t , which is the size of U_1. With this requirement, some subsets of U_2 whose sizes are greater than or equal to t and which are not in Γ can reveal m. Again this condition contradicts with the definition of Γ. Therefore, the dealer sets two polynomials shown in Equations (3.9) and (3.10). Since I_m is a piece of information relating to m and has the same function as a shadow used in Shamir's Scheme, and I_m is also used by the members of P to reveal m, the dealer can then set I_ms for the members of P using the following equations,

$$I_{mi} = ((x_i, Z_1(x_i)); \sum_{k=1}^{W} (x_k, Z_2(x_k))), \quad i \in U_1 \tag{3.11}$$

$$I_{mj} = (x_j, Z_2(x_j)), \qquad j \in U_2 \qquad (3.12)$$

where W is the weight of the members of U_1 over the members of U_2 (see Section 3.1 for the definition), x_i and x_k are arbitrary integer values assigned to each member i of U_1, x_j is an arbitrary integer value assigned to each member j of U_2, and $(x,Z_1(x))$ and $(x,Z_2(x))$ are units of I_m that are used for revealing m in the recovery phase. Equation (3.11) indicates that the first term, $(x_i,Z_1(x_i))$, is used only by the members of U_1 to reveal m using Equation (3.9) later in the recovery phase, and the second term is used by members of U_1, together with members of U_2, acting as members of the non-unique subsets to reveal m using Equation (3.10). Equation (3.12) is used by both members of U_2 and members of U_2, together with members of U_1, acting as members of the non-unique subsets, to reveal m using Equation (3.10).

Case II: If the total weight of all members of U_1 is greater than or equal to the number of members of U_2, $|U_1| * W \geq |U_2|$ and the total weight of all members less one of U_1 is less than the number of members of U_2, $(|U_1| - 1) * W < |U_2|$, then the dealer will set a random polynomial of degree $|U_2| - 1 = n - t - 1$ used for dividing m, where $|U_2| = n - t$.

$$Z_3(x) = d_0 + d_1 x + \ldots + d_{n-t-1} x^{n-t-1}, \text{ for } U_1 \text{ and } U_2 \qquad (3.13)$$

where $d_0 = m$, and the other d_is are arbitrary integer values. Since the degree of the polynomial is n - t -1, then at least $|U_2| = n - t$ units of I_m are needed to reveal m. In this case, all of the qualified subsets in Γ can satisfy this requirement. For U_2, the requirement is automatically satisfied because the polynomial is set based on the size of U_2. Since the non-unique subsets satisfy Equation (3.4), all of the subsets have at least (n - t) units of I_m so that they can reveal m using Equation (3.13). For U_1, since the total weight of U_1 is greater than or equal to (n - t) (meaning that it have at least (n - t) units of I_m), it can also reveal m. The dealer also sets I_ms for the members of P using the following equations,

$$I_{m\mathrm{i}} = \sum_{k=1}^{W} (x_k, Z_3(x_k)), \qquad i \in U_1 \qquad (3.14)$$

$$I_{m\mathrm{j}} = (x_j, Z_3(x_j)), \qquad j \in U_2 \qquad (3.15)$$

where W is the weight of the members of U_1 over the members of U_2 (see Section 3.1 for the definition), x_k is arbitrary integer value given to each member i of U_1, x_j is an arbitrary integer value given to each member j of U_2, and $(x, Z_3(x))$ represents one unit of I_m used for revealing m in the recovery phase.

Case III : If the total weight of all members less one of U_1 is greater than or equal to the number of members of U_2, $(|U_1| - 1) * W \geq |U_2|$, then the dealer will set two random polynomials of degrees $|U_1| - 1 = t - 1$ and $|U_2| - 1 = n - t - 1$, where $|U_1| = t$ and $|U_2| = n - t$. The dealer uses these polynomials for dividing m.

$$Z_4(x) = e_0 + e_1 x + \dots + e_{t-1} x^{t-1}, \quad \text{for } U_1 \qquad (3.16)$$

$$Z_5(x) = f_0 + f_1 x + \dots + f_{n-t-1} x^{n-t-1}, \text{ for } U_2 \text{ and other subsets} \qquad (3.17)$$

where $e_0 = f_0 = m$, and the other e_is and f_is are arbitrary integer values. The reason why the dealer sets two Z(x) is the same as in Case I, i.e., to keep the notion stated at the beginning of this section (Subsection 3.2.2) satisfied. If the dealer sets one Z(x) polynomial shown in Equation (3.16), then some subsets of

U_2 that are not in Γ, and whose sizes are greater than or equal to t, can reveal *m*. This situation violates the notion mentioned above. The same argument also applies if the dealer sets one Z(x) polynomial shown in Equation (3.17). In this case, some subsets of U_1 that are not in Γ, and whose units of I_m are greater than or equal to (n - t), can reveal *m*. This situation also violates the notion mentioned above. The dealer then sets I_ms for the members of *P* using the equations,

$$\mathrm{i} : I_{m\mathrm{i}} = ((x_i, Z_4(x_i)); \sum_{k=1}^{W} (x_k, Z_5(x_k))), \qquad \mathrm{i} \in U_1 \tag{3.18}$$

$$\mathrm{j} : I_{m\mathrm{j}} = (x_j, Z_5(x_j)), \qquad \mathrm{j} \in U_2 \tag{3.19}$$

where *W* is the weight of the members of U_1 over the members of U_2 (see Section 3.1 for the definition), x_i and x_k are arbitrary integer values given to every member i of U_1, x_j is an arbitrary integer value given to every member j of U_2, and $(x, Z_4(x))$ and $(x, Z_5(x))$ are units of I_m that are used for revealing *m* in the recovery phase. The difference between the second term of Equation (3.11) and the second term of Equation (3.18) is that in Equation (3.11) the values of x_k are different among members of U_1, while in Equation (3.18) the values of x_k are the same for members of U_1. This should be done to avoid the situation where some subsets of U_1 that are not in Γ can reveal *m* (as mentioned above when discussing the reason why the dealer sets two Z(x) polynomials). By setting the same I_m for each member of U_1, some subsets of U_1 can never reveal *m* using Equation (3.17).

As mentioned at the beginning of Section 3.1, there is a certain polynomial Y(x) as shown in Equation (3.1), that is used by the dealer and certain subsets of members of P in Γ called the qualified subsets. In the dealer site, Y(x) is used to obtain the unique coefficients of Y(x) as described in the creation phase (Subsection 3.2.1), while for the qualified subsets, Y(x) is used to determine the secret key K by evaluating Y(x) at one value of the set S of shadows (see Subsection 3.2.1) as shown in Equation (3.5). Therefore, the coefficients of Y(x) should be distributed to the members of P. In order to increase the difficulty for each member of P in guessing the secret key K, besides partitioning m into I_ms as discussed earlier, the proposed scheme requires that the dealer distribute the coefficients of Y(x) to members of P such that each member of P will miss at least two coefficients of Y(x). Furthermore, the proposed scheme also requires, as done on the prime number m, that only the qualified subsets in Γ can reconstruct the polynomial Y(x).

The set P consists of two unique subsets U_1 and U_2, and the degree of Y(x) is $r = (|U_1| + 1) * |U_2|$. This means that there are r coefficients of Y(x) (see Equation 3.1) and, thus by considering all other relating assumptions and notations discussed in Items (4) and (5) in Section 3.1, the dealer distributes the coefficients of Y(x) to the members of P according to the following basic steps.

1. For members of U_1:
   - The first q coefficients of Y(x), where q is an integer variable and $q = (r - |U_2|) = (|U_1| * |U_2|)$, should be given to the members of U_1 in such a way that for those coefficients, every member of U_1 can only have $(|U_1| - 1) * |U_2|$ items. The dealer distributes the coefficients so that for the i^{th} member of U_1 the dealer skips the i^{th}

$|U_2|$ = (n-t) items and gives the remaining items. For example, for the first member of U_1 the dealer omits the first $|U_2|$ = (n-t) items and gives the remaining items to the first member, and for the second member of U_1 the dealer omits the second $|U_2|$ = (n-t) items and gives the remaining items to the second member. The reason for doing this is to guarantee that the requirement is satisfied (the requirement that each member of U_1 misses at least two coefficients as mentioned above). Based on the assumption stated in Item (4) in Section 3.1, the minimum size of U_2 is 3. Therefore each member of U_1 will miss at least three coefficients of Y(x) and this condition satisfies the above requirement.

- The last $|U_2|$ (i.e., (r - q) = $|U_2|$) coefficients of Y(x) are given to all U_1 members. Since the distribution of the first q coefficients of Y(x) for each member of U_1 has satisfied the requirement, the last $|U_2|$ coefficients of Y(x) can be given to all members of U_1.

The following example is given in order to have a better idea about the distribution of the coefficients of Y(x) to members of U_1 satisfying the requirement. Suppose, for instance, that $|U_1|$ = 2, $|U_2|$ = 3, U_1 = {P_1,P_2}, and U_2 = {P_3,P_4,P_5}. By applying the above rules, the coefficients of Y(x) can be distributed to the members of U_1 as depicted in Table I.

TABLE I. AN EXAMPLE FOR THE DISTRIBUTION OF THE COFFFICIENTS OF Y(X) FOR MEMBERS OF THE FIRST UNIQUE SUBSET U_1

participant	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
P_1	X	X	X	Y	Y	Y	Y	Y	Y
P_2	Y	Y	Y	X	X	X	Y	Y	Y

(Legend: X = not given and Y = given)

2. For members of U_2:

- The last $|U_2|$ coefficients of Y(x) are distributed such that each member of U_2 can only have one of the coefficients. Applying this rule to the unique subset U_2, whose minimum size is 3, will give a result that satisfies the requirement. Besides that, this rule also gives a condition where the last $|U_2|$ coefficients of Y(x) can only be revealed if all of the members of U_2 pool their coefficients. For the example given above, the distribution of the last $|U_2|$ coefficients of Y(x) is depicted in Table II.

TABLE II. AN EXAMPLE FOR THE DISTRIBUTION OF THE LAST THREE COEFFICIENTS OF Y(X) FOR MEMBERS OF THE SECOND UNIQUES SUBSET U_2

participant	a_7	a_8	a_9
P_3	Y	X	X
P_4	X	Y	X
P_5	X	X	Y

- The first q coefficients of Y(x), where $q = (r - |U_2|) = (|U_1| * |U_2|)$, are distributed as follows. Since the first q coefficients of Y(x) have been distributed to the members of U_1 (applying the rule discussed in Item 1 above), the dealer should distribute this first q coefficients of Y(x) to members of U_2 such that when one or more members of U_1 ask some members from U_2 to pool their coefficients of Y(x), they can reconstruct the polynomial. In other words, the dealer should distribute this first q coefficients of Y(x) to members of U_2 such that all of the non-unique subsets in Γ can reconstruct the polynomial. The easiest way to do it is for the dealer to distribute all the coefficients to all members of U_2. In order to further implement the requirement, the following can be done. The good is to make it difficult for each member of P to guess the secret key K based on his/her own information by giving as few coefficients of Y(x) as possible to each member of P. We need to make sure not to harm the capability of the qualified subsets on Γ to reconstruct the polynomial Y(x). Thus the dealer distributes the first q coefficients of Y(x) to members of U_2 by considering the number of members in the non-unique subsets that are from U_2. To simplify the issue, the dealer limits the case into three conditions as follows.

- If $((|U_1| - 1) * W + 1) \geq |U_2|$ (in the event that Case I or Case II of partitioning m into I_ms occurs) <u>OR</u> if $W + 1 \geq |U_2|$ (in the event that Case III of partitioning m into I_ms occurs), then all of the first q coefficients are distributed to each member of U_2. This condition indicates that the number of members in the non-unique subsets that are from U_2 is only one. Therefore, in order for the non-unique subsets to be able to reconstruct the polynomial Y(x), the dealer distributes all of the first q coefficients to each member of U_2.
- Else if $((|U_1| - 1) * W + 2) \geq |U_2|$ (in the event that Case I or Case II of partitioning m into I_ms occurs) <u>OR</u> if $W + 2 \geq |U_2|$ (in the event that Case III of partitioning m into I_ms occurs), then the first q coefficients are distributed such that two members of U_2 can obtain all of the coefficients. This condition indicates that the number of members in the non-unique subsets that are from U_2 is two. Therefore, in order for the non-unique subsets to be able to reconstruct the polynomial, the dealer distributes the first q coefficients such

that two members of U_2 can obtain all of the coefficients. The way to distribute the first q coefficients is that for each $|U_2|$ coefficients, the dealer omits one coefficient and gives the other ($|U_2|$-1) coefficients to each member of U_2. The coefficients to be skipped among the members of U_2 are different from one to another. For the example given in Item 1 (p. 24) above, a distribution is depicted in Table III.

TABLE III. AN EXAMPLE FOR THE DISTRIBUTION OF THE FIRST SIX COEFFICIENTS OF Y(X) FOR MEMBERS OF THE SECOND UNIQUE SUBSET U_2 SATISFYING THE SECOND CONDITION

participant	a_1	a_2	a_3	a_4	a_5	a_6
P_3	Y	Y	X	X	Y	Y
P_4	Y	X	Y	Y	X	Y
P_5	X	Y	Y	Y	Y	X

- Else (when the number of members in the non-unique subsets who are from U_2 is three or more) the dealer distributes the first q coefficients of Y(x) such that the three or more members of U_2 can obtain all of the coefficients. Such a distribution needs additional work because without it, two members of U_2 can also obtain the coefficients. Before discussing the additional work, let p denote the number of members in the non-unique subsets that are from U_2 ($p \geq 3$). The additional work (as done on dividing the prime number *m* into I_ms discussed before) is to partition some of the coefficients of Y(x) into pieces of information Φs using a polynomial $\theta(x)$, which is similar to the Z(x) polynomial used to partition *m* into I_ms. The coefficients to be partitioned are those that members of U_1 do not have. Since p members of U_2 are able to obtain the first q coefficients, which include the coefficients to be partitioned, the degree of polynomial $\theta(x)$ used to partition the coefficients is p - 1.

$$\theta(x) = g_0 + g_1 x + \ldots + g_{p-1} x^{p-1}, \quad \text{for } U_2 \tag{3.20}$$

where g_0 = the coefficient to be partitioned, and the other g_is are arbitrary integer values. The dealer distributes the pieces of information, Φs, to members of U_2 using the following equation.

$$\Phi_{ij} = (x_{ij}, \theta(x_{ij})), \qquad i = a_1, a_2, \ldots, a_q \text{ and } j \in U_2 \tag{3.21}$$

where x_{ij} is an arbitrary integer value, i is the index of the coefficients to be partitioned, j is a member of U_2, and $q = r - |U_2| = (|U_1| * |U_2|)$. The dealer also distributes the remaining coefficients of the first q coefficients of Y(x) to members of U_2. In the example given in Item 1 above, a distribution of the coefficients is depicted in Table IV.

TABLE IV. AN EXAMPLE FOR THE DISTRIBUTION OF THE FIRST SIX COEFFICIENTS OF Y(X) FOR MEMBERS OF THE SECOND UNIQUE SUBSET U_2 SATISFYING THE THIRD CONDITION.

participant	a_1	a_2	a_3	a_4	a_5	a_6
P_3	Y	Y	Φ_{33}	Φ_{43}	Y	Y
P_4	Y	X	Φ_{34}	Φ_{44}	X	Y
P_5	X	Y	Φ_{35}	Φ_{45}	Y	X

(Legend: Φ_{ij} = a piece of information about a_i given to member j of U_2)

Using the basic steps above, the dealer gives each member P_i of P a set C_i of some coefficients of Y(x), such that the qualified subsets in Γ can obtain all coefficients of Y(x), which is in turn can be used to reconstruct the polynomial Y(x). Moreover, C_i, which is distributed to each member P_i of P, is set so that each member P_i will miss at least two coefficients of Y(x). This is done in order to increase the difficulty for each member P_i to guess the secret key K based on his/her own information.

The dealer should also give n out of r shadows in S to members of P because they will be used to determine the secret key K as described below in the recovery phase. In the dealer site, these n ($n = |P|$) shadows must be maintained since they will be used for obtaining the new coefficients of Y(x), whenever a new secret key K' is created.

The dealer groups all the information given to each member P_i of P as an information triplet, $T_i = <s_i, C_i, I_{mi}>$, where s_i is a shadow of K owned by P_i, C_i is a set of some of the coefficients of Y(x) given to P_i, and I_{mi} is P_i's piece of information about m. Finally, the dealer sends T_is to P through a secure channel and the dealer completes its work.

The process starting from the creation phase (Subsection 3.2.1) will cycle again each time a new secret K' needs to be generated. All parameters except the n original shadows must be recreated.

3.2.3 Recovery Phase

This phase begins whenever a subset in Γ, say D, wants to use K. In order to recover K, D should pool all its members' T_is (i.e., the shadow s_i, a set C_i of some coefficients of Y(x), and a piece of information I_{mi} about the prime number m) except the shadows. The proposed scheme requires that D, and any subset in Γ, perform two actions as follows.

a. D must find the prime number m by first pooling its members' I_ms to obtain all of the coefficients of an appropriate polynomial Z(x) created by the dealer, as shown by one of the Equations (3.9), (3.10), (3.13), (3.16), or (3.17) (see Subsection 3.2.2). Then the first coefficient of the polynomial is taken since this coefficient equals *m*. The way to obtain all of the coefficients of the polynomial is described below. Suppose that the approriate polynomial is shown by Equation (3.13).

$$Z_3(x) = d_0 + d_1 x + \dots + d_{n-t-1} x^{n-t-1}$$

Then members of D pool their I_ms $(x_1, Z_3(x_1))$, $(x_2, Z_3(x_2))$, ..., $(x_{n-t}, Z_3(x_{n-t}))$ to form the following equations.

$$\begin{aligned} Z_3(x_1) &= d_0 + d_1 x_1 + \dots + d_{n-t-1} x_1^{n-t-1} \\ Z_3(x_2) &= d_0 + d_1 x_2 + \dots + d_{n-t-1} x_2^{n-t-1} \\ &\vdots \\ Z_3(x_{n-t}) &= d_0 + d_1 x_{n-t} + \dots + d_{n-t-1} x_{n-t}^{n-t-1} \end{aligned} \tag{3.22}$$

The above equations can be expressed in the matrix form as

$$[\mathbf{X}][\mathbf{d}] = [\mathbf{Z}] \tag{3.23}$$

where $[\mathbf{X}]$ is the (n-t) by (n-t) integer values matrix, $[\mathbf{d}] = [\, d_0 \; d_1 \; \dots \; d_{n-t}]^T$, and $[\mathbf{Z}] = [Z_3(x_1) \; Z_3(x_2) \; \dots \; Z_3(x_{n-t})]^T$. The matrix $[\mathbf{d}]$, which represents the coefficients of $Z_3(x)$, can be obtained if the inverse of matrix $[\mathbf{X}]$, $[\mathbf{X}]^{-1}$, is known.

$$\begin{aligned} [\mathbf{X}]^{-1}[\mathbf{X}][\mathbf{d}] &= [\mathbf{X}]^{-1} [\mathbf{Z}] \\ [\mathbf{I}]\,[\mathbf{d}] &= [\mathbf{X}]^{-1} [\mathbf{Z}] \\ [\mathbf{d}] &= [\mathbf{X}]^{-1} [\mathbf{Z}] \end{aligned} \tag{3.24}$$

where [**I**] is the identity matrix and $[\mathbf{X}]^{-1}$ is determined using the second method of finding an inverse matrix described in Section 2.1. By taking the first element of [**d**], d_0, the prime number *m* can thus be revealed since this element equals *m*.

b. Next, D must obtain all coefficients of Y(x), as shown in Equation (3.1) (see Section 3.1), by pooling all its members' C_is and then using these coefficients, together with each individual member's shadow, to evaluate Y(x) at the member's shadow s_i.

$$Y(s_i) = a_1[(s_i) \bmod m] + a_2[(s_i^2) \bmod m] + \ldots + a_r[(s_i^r) \bmod m] = K,\ i \in D \quad (3.25)$$

where *m* is the prime number and $r = (|U_1| + 1) * |U_2|$.

It should be noted that the evaluation of Y(x) at $x = s_i$ must be done individually by D's members since s_i s are kept secret by the members. After evaluating Y(x), each member of D must show its result to mutually check the correct answer.

The secret K itself can be used as a key for a cryptographic system such as the key for DES, Vigenere, or Beaufort system [Simmons92] [Seberry89], or as a secret vault-lock combination [Blundo93].

After K has been used, all information used to recover K must be discarded, and the dealer should create a new secret K' and other information (i.e., *C*s and I_ms), and distribute these pieces of information to members of *P*.

3.3 Practical Example

This subsection discusses an example, which might occur in the real world, to illustrate the implementation of the proposed scheme.

Suppose there is a company with two managers A and B, and three senior staff members, C, D and E. The company has a locked vault X that can only be opened using a secret combination K. It is desired that K can be recovered and hence X can be opened, if the managers gather their knowledge about K or if one manager gathers his/her knowledge with the other two senior staff members' knowledge, or if the three senior staff members gather all their knowledge. The problem is how to construct a secret sharing scheme that satisfies the above requirement.

The set of participants in this example is $P = \{A,B,C,D,E\}$ and the set of qualified subsets of P, called access structure Γ, that can recover K is as given below.

$$\Gamma = \{\{A,B\},\{C,D,E\},\{A,C,D\},\{A,C,E\},\{A,D,E\},\{B,C,D\},\{B,C,E\},\{B,D,E\}\}$$

Based on the (t,n) threshold scheme's definition (see Chapter I and Section 2.2.3), it is clear that Shamir's scheme cannot be implemented for this example because Γ doest not contain all subsets whose sizes are t or more ($t \leq n$). The proposed scheme, on the other hand, can be used to solve the problem.

Based on the proposed scheme, the unique subsets U_1 and U_2 are (A,B) and (C,D,E), since $U_1 \cap U_2 = \varnothing$ and $|U_1| < |U_2|$. In the above example, one member of U_1 is capable of taking an action on behalf of one member of U_2, which means that the weight W of the members of U_1 over the members of U_2 is 1, $W = 1$ (see Section 3.1 for definitions). Since the size of the smallest subset in Γ is 2, $t = |U_1| = 2$, and that of the largest subset in Γ is 3, $|U_2| = 3$, so the value of r can be computed by using the following equation.

$$\begin{aligned} r &= (|U_1| + 1) * |U_2| \\ &= (2 + 1) * 3 = 9 \end{aligned}$$

To solve the problem, the proposed scheme works as follows. Firstly, the proposed scheme, the dealer in this case, selects K as the secret combination, and chooses a set S of 9 (since $r = 9$) distinct integer values used as shadows of K, 5 of which are given to P, $S =$

$\{ s_A, \ldots, s_E, s_1, \ldots, s_4 \}$. The dealer also chooses a prime number m and sets a polynomial whose degree is r = 9.

$$Y(x) = a_1 x + a_2 x^2 + \ldots + a_9 x^9 \qquad (3.26)$$

The coefficients of Y(x) are unique such that Y(x) always gives the same result when it is evaluated at the values of $S = \{ s_A, \ldots, s_E, s_1, \ldots, s_4 \}$.

$$a_1 [(s_A) \bmod m] + a_2 [(s_A^2) \bmod m] + \ldots + a_9 [(s_A^9) \bmod m] = K$$
$$a_1 [(s_B) \bmod m] + a_2 [(s_B^2) \bmod m] + \ldots + a_9 [(s_B^9) \bmod m] = K$$
$$\vdots$$
$$a_1 [(s_4) \bmod m] + a_2 [(s_4^2) \bmod m] + \ldots + a_9 [(s_4^9) \bmod m] = K$$

where *m* is a prime number and K is the secret key. Using the second method of finding an inverse matrix, as described in Section 2.1, and then applying the matrix operation shown in Equation (3.7), all of the coefficients of Y(x) can be determined.

Secondly, after obtaining the coefficients of Y(x), the dealer divides the prime number *m* into pieces of information I_ms. Since $|U_1| * W = 2 * 1 = 2 < |U_2| = 3$, the dealer sets two random polynomials whose degrees are $|U_1|-1 = 2-1$ and $|U_2|-1 = 3-1$. (see Case I in Subsection 3.1.2).

$$Z_1(x) = b_0 + b_1 x \qquad (3.27)$$

$$Z_2(x) = c_0 + c_1 x + c_2 x^2 \qquad (3.28)$$

where $b_0 = c_0 = m$, and the other b_is and c_is are arbitrary integer values. For each member in U_1 and U_2, the dealer gives I_ms using the following equations.

$$I_{mi} = ((x_i, Z_1(x_i)); \sum_{k=1}^{W} (x_k, Z_2(x_k))), \qquad i \in U_1$$

$$I_{mj} = (x_j, Z_2(x_j)), \qquad j \in U_2$$

so that

$$I_{m_A} = ((x_A,Z_1(x_A));(x_A,Z_2(x_A)))$$

$$I_{m_B} = ((x_B,Z_1(x_B));(x_B,Z_2(x_B)))$$

$$I_{m_C} = (x_C,Z_2(x_C))$$

$$I_{m_D} = (x_D,Z_2(x_D))$$

$$I_{m_E} = (x_E,Z_2(x_E))$$

The dealer also distributes the coefficients of Y(x) to the members of *P*. Since Case I in dividing *m* occurs and $((|U_1| - 1) * W + 2) \geq |U_2|$ (see Section 3.2.2), then the dealer distributes the coefficients such that two members of U_2 can obtain the first $r - |U_2|$ coefficients of Y(x). Such a distribution is shown in Table V below.

TABLE V. DISTRIBUTION OF THE COEFFICIENTS OF Y(X) FOR ALL PARTICIPANTS OF THE EXAMPLE

participant	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
A	X	X	X	Y	Y	Y	Y	Y	Y
B	Y	Y	Y	X	X	X	Y	Y	Y
C	Y	Y	X	Y	Y	X	Y	X	X
D	X	Y	Y	X	Y	Y	X	Y	X
E	Y	X	Y	Y	X	Y	X	X	Y

The dealer gives each member P_i of *P* a set C_i of some coefficients of Y(x).

$$C_A = \{a_4, a_5, a_6, a_7, a_8, a_9\}$$

$$C_B = \{a_1, a_2, a_3, a_7, a_8, a_9\}$$

$$C_C = \{a_1, a_2, a_4, a_5, a_7\}$$

$$C_D = \{a_2, a_3, a_5, a_6, a_8\}$$

$$C_E = \{a_1, a_3, a_4, a_6, a_9\}$$

The dealer groups all of these pieces of information (i.e., s_i, C_i, and I_{m_i}) into an information triplet T_i.

$$T_A = <s_A, C_A, I_{m_A}>$$

$$T_B = <s_B, C_B, I_{m_B}>$$

$$T_C = <s_C, C_C, I_{m_C}>$$

$$T_D = <s_D, C_D, I_{m_D}>$$

$$T_E = <s_E, C_E, I_{m_E}>$$

The dealer then sends these T_is to the members of P and the dealer's work is completed.

Thirdly, if a subset of P in Γ, say $\alpha = \{A,C,D\}$, wants to open the locked vault X, it must get the secret combination K first. To have K, members of α must do two things.

a. First, the members must pool their I_ms to reconstruct the polynomial Z(x), in this case $Z_2(x)$, and then find the prime number m.

$$\begin{aligned} Z_2(x_A) &= c_0 + c_1 x_A + c_2 x_A^2 \\ Z_2(x_C) &= c_0 + c_1 x_C + c_2 x_C^2 \\ Z_2(x_D) &= c_0 + c_1 x_D + c_2 x_D^2 \end{aligned} \tag{3.29}$$

Using matrix operations described in Equations (3.22), (3.23), and (3.24), the coefficients of $Z_2(x)$ (i.e., c_0, c_1, and c_2) can be determined, and thus the prime number m can be obtained since $c_0 = m$.

b. Second, the members must pool their C_is to be able to get all of the coefficients of Y(x). In this case C_A, C_C, and C_D should be pooled so that the coefficients of Y(x) (i.e., a_1, a_2,

..., and a_9) can be obtained, and Y(x) is evaluated at each individual member's shadow s_i to recover K.

$$Y(s_i) = a_1[(s_i)\bmod m] + a_2[(s_i^2)\bmod m] + \ldots + a_9[(s_i^9)\bmod m] = K$$

where m is the prime number and i = A, C, or D.

Having the secret combination K, α can then open the locked vault X. After K has been used, all information utilized to recover K, except the shadows owned by the members of P, must be discarded. The dealer then starts working again by selecting a new secret K', processing it and sending new triplets T_is to the members of P. The difference between the new triplets and the old ones is that the new triplets are sent without the shadows (s_is), since the shadows have been sent to members of P by the dealer before.

$$T_i = <C_i, I_{mi}>, \quad i \in P$$

The next chapter discusses two issues relating to the proposed scheme. First, it discusses the analysis of the proposed scheme, which is divided into two aspects: security and recycleability. Second, it gives a comparison between the proposed scheme and Shamir's scheme.

CHAPTER IV

ANALYSIS AND COMPARISON WITH SHAMIR'S SCHEME

4.1 Analysis of the Proposed Scheme

This section discusses two important aspects of the proposed scheme, which was described in Chapter III. There aspects are security and recycleability. The security aspect concerns the protection of the secret key K against the unqualified subsets of participants that may try to reveal it. Recycleability means that the n original shadows that have been given to the n participants can be used again to recover a new secret key K'.

4.1.1 Security

A secret sharing scheme is said to be a secure scheme if it guarantees that any collaboration of unqualified participants cannot determine the secret key (information) [Simmons92]. The proposed scheme described in Chapter III has indeed such a property. The distribution of the required information (for obtaining the secret key K) in the proposed scheme is performed in such a way that any unqualified subset of the participants that neither is in Γ nor is a superset of the qualified subset in Γ can determine the secret key K. The proposed scheme satisfies the property by partitioning the prime number m (used both by the dealer to determine the coefficients of Y(x) as shown in Equations (3.1) and (3.5), and by a participant to derive the secret key K as shown in Equation (3.25) in the last chapter) into pieces of information I_m, and by distributing these I_ms to participants such that only the qualified subsets of participants in Γ can reveal m by pooling their I_ms. The way to partition the prime number m into I_ms was discussed in Subsection 3.2.2 of the Chapter III. In addition to partitioning m into I_ms, the proposed scheme also distributes a set C_i of some coefficients of the polynomial Y(x), used as a template to determine the secret key K, to each participant such that only the qualified subsets in Γ can reconstruct the polynomial Y(x) by pooling the C_is they have. Furthermore, in receiving C_i, each participant will miss at least two coefficients of Y(x). This is done in order to increase the difficulty for each participant that tries to guess the secret key K based on each participant's own information.

By applying the requirement that only the qualified subsets in Γ can reveal m and reconstruct the polynomial Y(x), it will be difficult for any unqualified subset of participants to determine the secret key K because the two types of information needed for obtaining K are unknown (i.e., the prime number m and one or more coefficients of Y(x)).

To have a better understanding of how the proposed scheme satisfies the property, recall the practical example given in Section 3.3 of the previous chapter. In that example, a collaboration among the members of an unqualified subset of the participants, say δ = {A,D}, whose members are from the unique subsets U_1 and U_2, will not determine the secret key K because of the following three reasons.

1. Not all of the coefficients of Y(x) shown in Equation (3.26) (see Section 3.3) can be obtained if the members of δ pool their C_is. In this case, the coefficient a_3 of the equation is missing if A and D gather their C_A and C_D as shown in Table V in Section 3.3.

2. A and D cannot reveal the prime number m by pooling their I_ms since they have only two out of the three I_ms. These three items should be possessed by the members of δ in order to obtain m as shown in Equations (3.28) and (3.29).

3. Since m and one coefficient of Y(x) shown in Equation (3.26), a_3, cannot be revealed, the members of δ cannot determine K. Guessing K is very difficult because they should first guess the values of the two unknown information: m and a_3.

4.1.2 Recycleability

The proposed scheme has the recycleability property in the sense that the n original shadows, which have been distributed to the n participants for the first time, can be used many times independently of the creation of the secret key (information). This feature will ease the dealer's work since generating and distributing new shadows, when a new secret key K' is created, is time consuming.

Recycleability is one of the two important characteristics of the proposed scheme that differentiate it from a (t,n) threshold scheme, since a (t,n) threshold scheme does not have this property. The other characteristic is that the proposed scheme supports a general access structure secret sharing scheme.

4.2 Comparison with Shamir's Scheme

Compared to Shamir's scheme, the proposed scheme has a number of advantages as listed below.

1. The proposed scheme is designed for general access structure. The example given in Section 3.3 indicates that the proposed scheme can be used to solve a general access structure secret sharing problem, while Shamir's scheme cannot be implemented for the

problem because, in fact, the (t,n) threshold scheme is only a special case of the general access structure secret sharing scheme as also stated in [Blundo93].

2. In Shamir's scheme, the value of the secret key K must be less than the value of the prime number p (see Subsection 2.2.2), while in the proposed scheme, the value of K is independent of that of the prime number m. This will increas the difficulty in guessing K because K can be greater than, less than, or even equal to the prime number m.
3. Preserving the shadows used for recovering K is another advantage since they can be used many times independently of the creation of the secret K .

The proposed scheme, however, has two disadvantages. First, the amount of information to be sent to participants is greater than the amount of information needed to be sent to participants in Shamir's scheme. This condition may cause a problem in keeping the information (i.e. $T_i = <s_i, C_i, I_{mi}>$) secret since it will be very difficult (or even impossible) for each participant to memorize the information. One possible solution that can be used to prevent the information from being stolen or changed is to put the information into a card, and to give each participant such a card.

The second disadvantage of the proposed scheme is that the size of the first unique subset is limited only to two members. This limitation exists because of the nature of the distribution of the coefficients of Y(x), which requires (as done on the prime number m) that only the qualified subsets in Γ can reconstruct the polynomial Y(x). If the size of U_1 is increased to be three or more, the distribution of C_is using the steps described in Subsection 3.2.2 will result in a condition where there is one or more unqualified subsets of participants that can reconstruct the polynomial Y(x). This will make the unqualified subsets (in guessing the secret key K) concentrate only on, althought it is still difficult to do that, finding the correct value of the prime number *m*, which is left unknown.

The next chapter describes an example of the implementation of the proposed scheme on a network program. The program is called Secret Conference. Secret Conference is one form of a conference program, which allows multiple users to converse secretly by encrypting and decrypting the data to be communicated. The data is encrypted and decrypted using a secret communication key that is produced by applying the proposed scheme.

CHAPTER V

SECRET CONFERENCE: AN IMPLEMENTATION OF THE PROPOSED SCHEME IN A NETWORK PROGRAM

5.1 Program Description

This chapter discusses a network program that is used as an example of the implementation of the proposed scheme described in Chapter III. The program, called Secret Conference, was created by the author as a form of Conference Calling, which was one of the several projects of the Computer Networking course (COMSC 4283) offered in Fall 1994 at the Computer Science Department of Oklahoma State University. Conference Calling is a network program that allows multiple users to communicate with each other by setting a conference. Conference Calling uses a client-server model where the server is responsible for directing communication flows among clients who join the conference. Secret Conference modifies the conference so that it runs secretly by encrypting and decrypting the data to be communicated among the clients. The encryption and decryption are carried out using a secret key K created by the proposed scheme. In Secret Conference, the server acts as the dealer whereas each client functions as a participant as described in Chapter III. In addition, Secret Conference requires that:

A. the conference be set only by qualified subsets of participants as described in the distribution phase in Chapter III.

B. the qualified subsets of participants who set or join a conference determine the secret key K using the steps described in the recovery phase in Chapter III before the conference takes place (Secret Conference, on the client's part, is designed to determine the secret key automatically).

C. the qualified subsets of participants who set or join a conference encrypt the data to be sent or decrypt the incoming data using the secret key K in order to achieve a secure and secret conference.

Secret Conference uses other cryptographic systems: Vigenere, Beaufort, and Variant Beaufort [Seberry89], in conjunction with the proposed scheme, for encrypting and decrypting the communicated data using the secret key K generated by the proposed scheme.

As the dealer, the server is responsible for providing all the information needed by participants (or clients for short). The server is also responsible for:

- setting, together with the corresponding client, a secret key upon receiving a connection request made by a client. This secret key is used for secret communication between the server and the client. Every client possesses a different secret key.
- distributing the required information to all clients if either all connections to all clients are set or a conference is ended.
- serving a request invoked by a client.
- determining all of the qualified subsets of clients (in Γ) that are eligible for establishing a conference.
- granting or rejecting, based on information in Γ, a conference request sent by a client.
- directing communication flows among the clients.

A client does several things. First, it makes a connection request to the server. If the request fails, the client is given two options. Either it can send the same request to the server again until the request is successful, or it can decide to quit. Second, if the connection request succeeds, the client can do the following.

- The client may invoke any available request for obtaining the corresponding service provided by the server.
- By following the requirements (A) to (C) above, the client may send a conference request and start comunicating with other clients who have joined the conference.

5.2 Program Design

This section discusses several issues in designing Secret Conference. First, it gives a brief information about network connections used in Secret Conference for communication between a the server and a client as well as among clients. Second, it discusses the program structure of Secret Conference.

5.2.1 Network Protocol

Secret Conference uses a connection-oriented network protocol, provided by Berkeley Sockets, for communication either between the server and a client or among clients through the server. Figure 2 describes a typical scenario of the protocol for a client-server model.

The connection-oriented protocol requires that the server and the client establish a logical connection with each other before communication begins, and the server should run first before executing the client(s) [Stevens90].

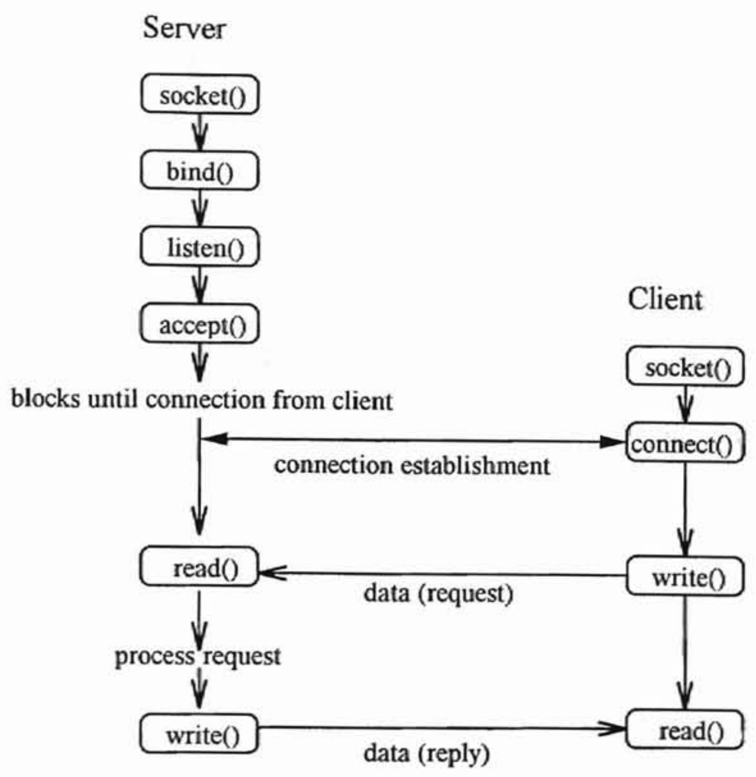


Figure 2. Socket system calls for a client-server model using the connection-oriented protocol (Source: [Stevens90])

5.2.2 Program Structure

The Secret Conference program is divided into three parts: define.h, Server.c, and Client.c. The define.h module is a header file for Secret Conference program, while Server.c and Client.c simulate a server (the dealer) and a client (participant), respectively. These three modules are described below.

The define.h module consists of all items that are shared by Server.c and Client.c. These items are classified into three types.

- All C library header files including network header files and math header files.
- All symbolic constants. These constants are used by Server.c and Client.c
- Procedures used by both Server.c and Client.c.

* remainder procedure: This procedure is used for obtaining the final result of modular computation of the shadows of the secret key as shown in Equation (3.5) in Chapter III.

* encrypt procedure: This procedure is used for encrypting the data to be sent.

* decrypt procedure: This procedure is used for decrypting the incoming data.

The Server.c module is designed to simulate a server (or the dealer in the proposed scheme). The function of Server.c has been discussed in Section 5.1 of this chapter. Server.c consists of 12 procedures, including the main procedure, which are described below.

1. generator procedure: This procedure functions as a random number generator.
2. matrix operation procedure: This procedure is used for determining the inverse matrix of the shadow matrix shown in Equation (3.7) in Chapter III.
3. create_prime procedure: This procedure creates all prime numbers, including m.
4. power_degree procedure: This procedure obtains the degree value of Y(x) shown in Equation (3.1) in Chapter III.
5. creation_phase procedure: This procedure selects and processes all items needed by clients. This procedure is called if either all clients have established their connections to the server or a conference has ended.
6. create_Z_polynomial procedure: This procedure creates one or two Z(x) polynomials.
7. divide_PRIME procedure: This procedure divides a prime number m into I_ms using the Z(x) polynomial(s) created by create_Z_polynomial procedure.
8. distribution_phase procedure: This procedure distributes all information created by creation_phase procedure to all clients. This procedure is called once all clients have established their connections to the server or a conference has ended.
9. info_exchange procedure: This procedure performs two important functions. First, it allows the server, upon receiving a connection request made by a client, to set, together with the corresponding client, a unique secret key used for secure communication. Second, it allows the server and the client to exchange their information using the secret key.

10. confirm procedure: This procedure allows the server to grant or reject a conference request sent by a client.

11. serve_request procedure: This procedure serves any request invoked by a client or directs communication flows among the clients who join a conference.

12. main procedure: This procedure performs two functions. First, it initializes all variables used by other procedures and for communicating with clients. Second, it applies all Socket system calls depicted in Figure 2 except the write() system call, since it is used in serve_request procedure.

The Client.c module consists of seven procedures, including main procedure, which are described below.

1. obtain_value procedure: This procedure obtains certain values, such as the prime number m and the cofficients of Y(x) to be partitioned, using matrix operations

2. distribute procedure: This procedure processes and saves all of the important information sent by the server such as the shadows, some coefficients of Y(x), and the I_ms.

3. info_display procedure: This procedure displays information about the client on stdout (screen).

4. process_reply procedure: This procedure performs two important functions. First, it processes any reply or the secret conference data sent by the server. Second, it receives other information about the coefficients of Y(x) and I_ms sent by the other client(s) through the server and processes the information such that the secret key used for the secret conference is determined.

5. info_exchange procedure: This procedure is the same as info_exchange procedure in Server.c.

6. help procedure: This procedure displays information about Secret Conference on stdout (screen).

7. main procedure: This procedure performs three fuctions. First, it initializes all variables used by other procedures or used for communicating with the server. Second, it sends a connection request to the server. Third, it processes incoming data either from stdin (keyboard) or from the server; if it is from the server, it calls the process_reply procedure.

The complete program is given in PROGRAM LISTING (APPENDIX C).

5.3 Running Secret Conference

Secret Conference consists of two programs: Server and Client. It is required that Server be executed before Client by issuing the following command.

```
Server n t w &
```

where n, t, and w are integer values, n is the total number of Clients (participants) who made connections to Server, t is the size of the first unique subset U_1, w is the weight of the members of U_1 over the members of U_2 (see Section 3.1 for definitions), and & indicates that the server is executed silently (in the background).

To run the Client, the following command should be issued.

```
Client Name Level
```

where Name is the client's name and Level is an integer value (1 or 2) referring to the index of the unique subset to which the client belongs. Client Nur 1, for instance, means that the client's name is Nur and the client is a member of the first unique subset U_1. Since the server part of Secret Conference is made to work passively (meaning that it produces no important message or prompts), the expected performance of Secret Conference is due to the output of the client part, which can be briefly described below.

1. When the client part is invoked, say Client Nur 1 as described above for instance, one of two things can happen.
   a) First, it may give a message indicating that a connection request to the server fails. In this case, it dispalys two options: quit or try again until the request succeeds and proceeds to (b).

   b) Second, it may display about one page worth of information about Secret Conference. It describes two types of commands: the request commands and the conference command.

2. It gives a message indicating that another client has made a connection to the server.
3. If all clients have made connections to the server, it displays a message that the server has sent all the information needed to determine the secret key K (required for establishing a conference and for communications among the clients who join the conference). At this stage, the client Nur, as well as any other client, can send a conference request to the server.
4. If the client Nur, or another client, sends a conference request satisfying the requirements (A) to (C) as stated in Section 5.1, and the server grants the request, then a message indicating that the conference may begin is displayed. After receiving such a reply from the server, the client Nur can communicate to other clients who have either participated in setting the conference or joined the conference later.

To achieve a secure and secret conference, a client should give an encryption type (Vigenere, Beaufort, or Variant Beaufort) each time it sends data to the other clients. The encryption and decryption are carried out using the secret key K. The commands used in the program (Secret Conference) are given in the User Guide for Secret Conference (APPENDIX B).

5.4 Analysis

Secret Conference was run under the DYNIX/ptx operating system on a Sequent S/81. It was initially tested for running three clients that were divided into two unique subsets. The first unique subset consisted of one client whose weight is one over the members of the second unique subset. In this case, since there are three clients that should be run, at least four processes must be created to accommodate the clients and the server by using either the X Window System or the *screen* application [Weigert91]. Running Secret Conference for three clients gives the expected output as described in Section 5.3.

Secret Conference was also tested for more than three clients. It was tested for running five clients (two clients belonging to the first unique subset and the weight of one) and, it was tested for running seven clients (three of which belonging to the first unique subset and the weight being one). Both tests produced the correct output as expected.

CHAPTER VI

SUMMARY AND FUTURE WORK

6.1 Summary

The idea of dividing a secret information into several pieces, called shadows, in such a way that a specified number of shadows must be combined to determine the secret information was proposed independently by Blakley and Shamir [Blakley79] [Shamir79]. If there are n shadows that are given to n participants and at least t ($t \leq n$) of them are required in order to obtain the secret information, then the scheme is called a (t,n) threshold or secret sharing scheme.

A number of secret sharing schemes have been developed implementing the notion that pooling t or more shadows can make the secret information determinable, while gaining t - 1 or fewer shadows leaves the secret information unknown (in the sense that all of its possible values are equally likely). Other researchers have developed the scheme using a general access structure. A general access structure is defined as a way to divide the secret information among a set of n participants such that only qualified subsets of the participants can recover the secret information, while those that are unqualified cannot recover it.

Secret sharing scheme is useful in information security, in multi-user network cryptographic systems, in opening a vault lock or a safety deposit box, or even in any controlled action which requires the presence of several persons in order to initiate an action.

Secret sharing scheme, however, has a drawback. Once the secret information is determined, all of the shadows, including those that are not involved in the recovery process, become useless. Consequently, when a new secret information is created, all participants should be given the corresponding shadows.

The purpose of this thesis was to introduce a new and different approach (scheme) that can remedy the drawback by maintaining all of the original shadows that have been distributed to the participants so that these shadows can still be used many times independently of the creation of the secret information. This feature is called recycleability of the shadows. Another feature of the proposed scheme is that it can be used as a general access structure.

In the proposed scheme, a set P of n participants is grouped into two unique subsets U_1 and U_2, where $U_1 \cap U_2 = \varnothing$ and $|U_1| + |U_2| = n$. To determine the secret key K, the proposed scheme creates a certain polynomial Y(x) whose degree r is $(|U_1| + 1) * |U_2|$.

$$Y(x) = a_1 x + a_2 x^2 + \dots + a_r x^r$$

The proposed scheme also selects a prime number m and r integer values, which function as the shadows of the secret key K. Of the r shadows, n are given to the participants. The n shadows are used by the participants to obtain K by evaluating the polynomial Y(x) at the values of the n shadows.

$$a_1 [(s) \bmod m] + a_2 [(s^2) \bmod m] + \dots + a_r [(s^r) \bmod m] = K \text{ where s is a shadow}$$

The n shadows are maintained so that they can be used repeatedly when a new secret key K' is created. This is how the proposed scheme maintains the recycleability property. To support the general access structure Γ, the proposed scheme performs two functions.

1. It divides/partitions the prime number m into pieces of information, I_ms, relating to m and distributes the I_ms to the participants.

2. It distributes a set C_i of some coefficients of Y(x) to each participant P_i of P. The reason for distributing C_i to each member P_i is to guarantee that each member P_i misses at least two coefficients of Y(x) so that it increases the difficulty in guessing the secret key K based on each member's information.

The partitioning and distribution of I_ms and C_i is carried out such that only the qualified subsets of participants in Γ can reveal the prime number m and reconstruct the polynomial Y(x).

The work of the proposed scheme is divided into three phases: creation, distribtion, and recovery. The first two phases are carried out in the dealer, a special and trusted participant. The dealer is responsible for:

1. creating the polynomial Y(x), the prime number m, the secret key K, and the set of integer values used as shadows of K.

2. partitioning m into I_ms.

3. distributing n shadows, the I_ms, and the C_is to the members of P.

The recovery phase begins when a subset of participants wants to recover the secret key K using all of the information distributed by the dealer.

The proposed scheme can be applied in the real world. One such applications was implemented in a network program called Secret Conference. Secret Conference is a network communication program that allows multiple users to converse secretly by encrypting and decrypting the data to be communicated. The data is encrypted and decrypted using a secret communication key K generated by the proposed scheme and other cryptographic systems (i.e., Vigenere, Beaufort, or Variant Beaufort [Seberry89]). Secret Conference uses the client-server model. The server, who acts as the dealer, is responsible for providing all of the information (which includes all of the information needed for obtaining the secret key K) required by the clients to establish a conference. In Secret Conference, a subset of the clients, which is a qualified subset on Γ, may start making a conference if it can obtain the secret comunication key K created by the server. The subset can subsequently use K for encrypting and decrypting the data communicated among the members of the subset.

6.2 Future Work

The proposed scheme developed in this thesis has a limitation. The size of the first unique subset U_1 is limited only to two members. This limitation emerged because of the method used to distribute the coefficients of Y(x). This method requires (as done on the prime number m) that only the qualified subsets in Γ can reconstruct the polynomial Y(x). If the size of U_1 is increased to be three or more, the distribution sets C_is of some coefficients of Y(x) (using the steps described in Subsection 3.2.2) could lead to a condition where there is one or more unqualified subsets of participants that can reconstruct the polynomial Y(x). This makes the unqualified subsets (in guessing the secret key K) focus on obtaining the correct value of only one type of information that is left unknown: the prime number m, even though finding the true value of m is still difficult.

This restriction opens up several avenues for further work. These include removing the restriction such that the proposed scheme can be used for any size of the first unique subset, or finding other schemes that preserve the recycleable shadow feature and support a general access structure.

REFERENCES

[Anton81] H. Anton, *Elementary Linear Algebra*, John Wiley & Sons, New York, NY, 1981.

[Blakley79] G.R. Blakley, "Safeguarding Cryptographic Keys", *Proc. of AFIPS National Computer Conference*, vol. 48, New York, NY, pp. 313-317, 1979.

[Blundo93] C. Blundo, A. De Santis, L. Gargano, and U. Vaccaro, "On the Information of Secret Sharing Scheme", *Advances in Cryptology - Crypto'92*, Springer-Verlag, Berlin, pp. 148-167, 1993.

[Brickell90] E.F. Brickell and D.R. Stinson, "The Detection of Cheaters in Threshold Scheme", *Advances in Crytology - Crypto'88,* Springer-Verlag, Berlin, pp. 148-167, 1990.

[Denning82] D.E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.

[Harn93] Lein Harn and Hung-Yung Lin, "An l-Span Generalized Secret Sharing Scheme", *Advances in Cryptology - Crypto'92*, Springer-Verlag, Berlin, pp. 558-565,1993.

[Hershey86] John E. Hershey and R.K. Rao Yarlagadda, *Data Transportation and Protection*, Plenum Press, New York, NY, 1986

[Jackson94] Wein-Ai Jackson, K. Martin, and C.M. O'Keefe, "Multisecret Threshold Scheme", *Advances in Cryptology - Crypto'93*, Springer-Verlag, Berlin, pp. 126-135, 1994.

[Rompel90] J. Rompel, "One-Way Functions Are Necessary and Sufficient for Secure Signatures", *Proc. of the 22nd ACM Symp. on Theory of Computing*, Baltimore, MD, pp. 387-394, 1990.

[Seberry89] Jennifer Seberry and Josef Pieprzyk, *Cryptography: An Introduction to Computer Security*, Prentice-Hall, Sidney, 1989.

[Shamir79] A. Shamir, "How to Share a Secret", *Comm. of the ACM*, vol. 22, pp. 612-613, 1979.

[Shannon49] C.E. Shannon,"Communication Theory of Secrecy System", *Bell Syst. Tech. Journal*, vol. 26, pp. 656-715, 1949.

[Simmons89] G.J. Simmons, "Robust Shared Secret Scheme", *Congressus Numerantium*, vol. 68, pp. 215-248,1989.

[Simmons92] G.J. Simmons, *An Introduction to Shared Secret and/or Shared Control Schemes and Their Application*, Contemporary Cryptology, The Science of Information Integrity, IEEE Press, New York, NY, pp. 441-497, 1992.

[Stevens90] W. Richard Stevens, *UNIX® Network Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[Stinson93] D.R. Stinson, "New General Lower Bounds on the Information Rate of Secret Sharing Scheme", *Advances in Cryptology - Crypto'92*, Springer-Verlag, Berlin, pp. 168-182, 1993.

[Weigert91] Juergen Weigert and Michael Schroeder, *Screen*, Free Software Foundation, Inc., Cambridge, MA, Copyright(c) 1991.

APPENDICES

APPENDIX A

GLOSSARY AND TRADEMARK INFORMATION

GLOSSARY

Access Structure: A collection of all subsets of participants that can recover the secret information.

Client-Server model: A model where one process is responsible for providing some facilities to the other processes. The former process is known as a server process and the latter are called client processes.

Connection-Oriented Network Protocol: A network protocol that requires two processes or applications establish a (logical) connection with each other before data can be sent back and forward.

Cypher Data: The scrambled data obtained from plain data.

Dealer: A special and trusted participant that selects the secret information and gives the other participants all the information that is needed to derive the secret.

Decryption: A method to determine plain data from cypher data.

Encryption: A method to scramble plain data such that one cannot obtain any information from the scrambled data.

Euclidean Space (E^n): A geometric model on which the position of an object can be determined; e.g., the position of an object can be expressed in three-dimensional geometric model (X, Y, and Z axes) called E^3.

General Access Structure: Access structure realizing the notion that only qualified subsets of participants can recover a secret.

Hyperplane: A plane obtained as a result of projection of an object on projective space.

Identity Matrix: A matrix that has 1s in its diagonal entries and 0s in all other entries.

Linear System or System of Linear Equations: A set of n linear equations.

Monotone: An access structure characteristic where, if there is a subset A of the participants that is a superset of a qualified subset B in the access structure, then A can also reveal the secret.

Network Protocol: A set of (computer) network rules and conventions used for communication among participants

Participants: A set *P* of people or processes that have the right to recover the shared secret information.

Plain Data: Meaningful information to be sent back and forward among participants.

Projective Space (PG): A geometric model to locate an object based on the projections of the object.

Secret Sharing Scheme: A method to protect a shared secret information among a set of participants such that certain subsets of the participants can derive the secret.

Shadow: A piece of information relating to the secret information to be shared.

Socket: A port address used as a channel for communication between two processes.

Threshold Scheme: A secret sharing scheme that contains all subsets of participants of size at least t.

Unique Subset: A subset of *P* whose members are from same level, which is different from the level of the rest of the participants in *P*, whose members are not the members of the unique subset.

Weight: The capability of one member in a unique subset to take an action on behalf of one or two of the other participants.

TRADEMARK INFORMATION

Berkeley Sockets is a registered trademark of the University of California at Berkeley.

Sequent S/81 is a registered trademark of the Sequent Computer System, Inc.

APPENDIX B

USER GUIDE FOR SECRET CONFERENCE

Secret Conference is a network program that allows multiple users to hold a conference secretly by implementing the proposed scheme described in Chapter III. In order to achieve a secure and secret conference, the Secret Conference program uses other cryptographic systems called Vigenere, Beaufort, and Beaufort Variant. The cryptographic systems use the secret key K generated by the proposed scheme for encrypting and decrypting the data to be communicated. Secret Conference consists of two programs: Server and Client. Server runs silently and should be run before Client is executed. The command for running Server is *Server n t w &*, where *n* is the total number of clients that are divided into two groups, *t* is the size of the first group, *w* is the weight of the first group's members over the second group's members, and *&* indicates that the server runs in the background. Since the total number of clients is *n*, Client should be executed *n* times in *n* different processes. The command used for running Client is *Client Name Level*, where *Name* is the client's name and *Level* is an integer value (1 or 2) indicating the group to which the client belongs.

Suppose that Secret Conference is used for running five users, namely *Nur*, *Ina*, *Dija*, *Abdu*, and *Alice*. These users are divided into two groups. The first group has two members who are *Nur* and *Ina*. Server is invoked by typing: *Server 5 2 1 &*. Client is invoked three times: *Client Nur 1*, *Client Ina 1*, *Client Dija 2*, *Client Abdu 2*, and *Client Alice 2*, in five different processes. Several issues can be pointed out when Secret Conference runs these five clients.

a) When the *Client Nur 1* command is issued and the client successfully makes a connection to Server, a number of messages are displayed on stdout (i.e., the screen) as depicted in Figure 3. The same thing also happens to other clients if they are created as additions to the client *Nur* and other clients who have made connections to Server. In such a case, the Server sends a message indicating that another client is active, as shown in Figure 4.

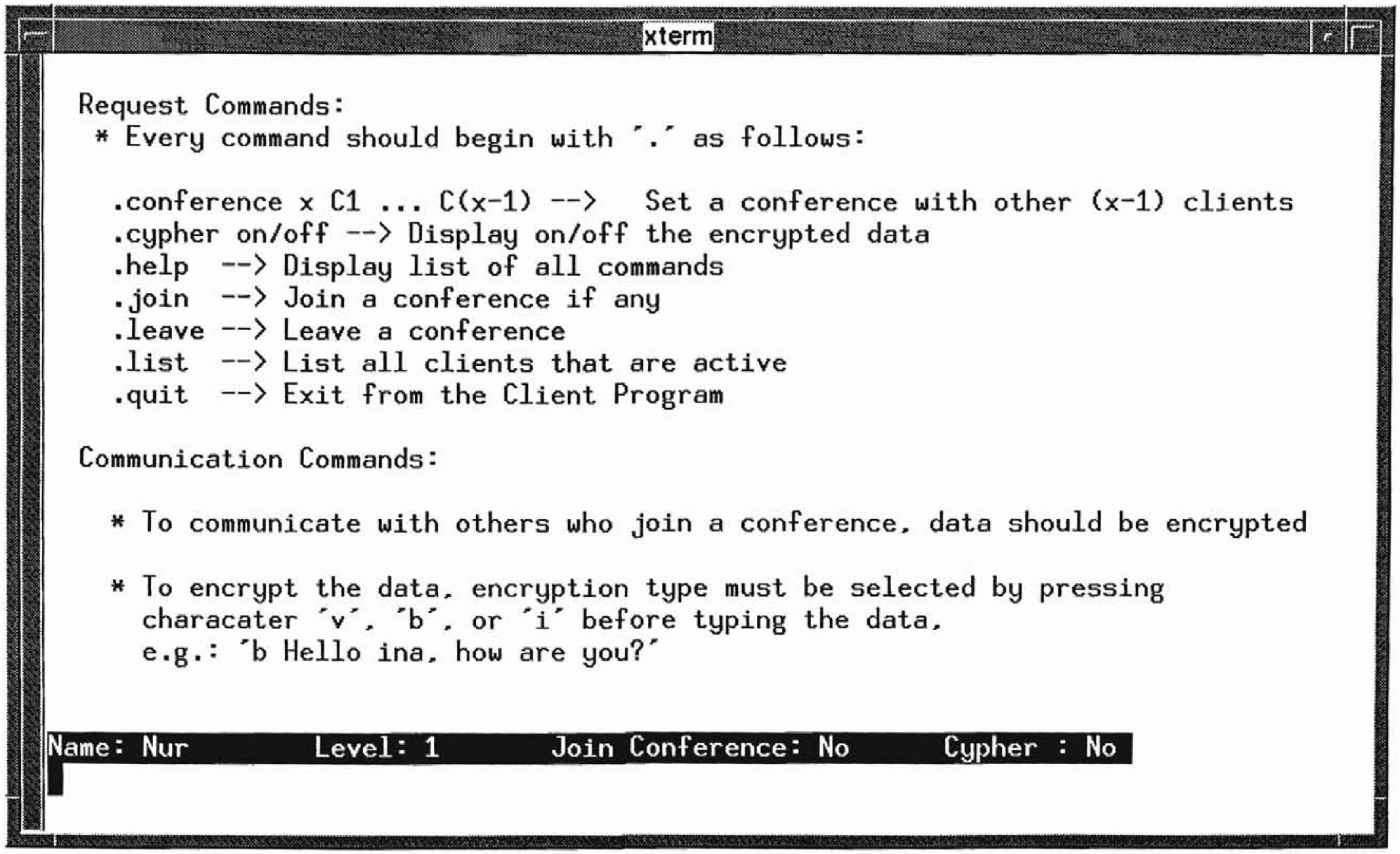
b) If all clients have made connections to the Server, the Server then sends a message to them notifying them that the Server has distributed all information required for establishing a secret and secure conference. Figure 5 shows the message.

In Secret Conference, commands issued by the clients are divided into two types: request commands and a command used in the conference. To differentiate the first type of commands from the second type, every request comand should begin with '.'. There are seven request commands used in Secret Conference as shown in Figure 3.

1) *.conference* x C_1 C_2 ... C_{x-1}. This command is used when a client wants to set a conference. x is an integer value referring to the number of clients, including the client sending this request wanting to establish a conference, and C_1 C_2 ... C_{x-1} are the names of the other (x-1) clients. In the above example, client *Nur* may issue a conference request by invoking a command: *.conference 2 Ina*; if the Server grants the request, the Server then sends a message to all clients informing them that a conference may begin. Figures 6 and 7 show the request and the successful message.

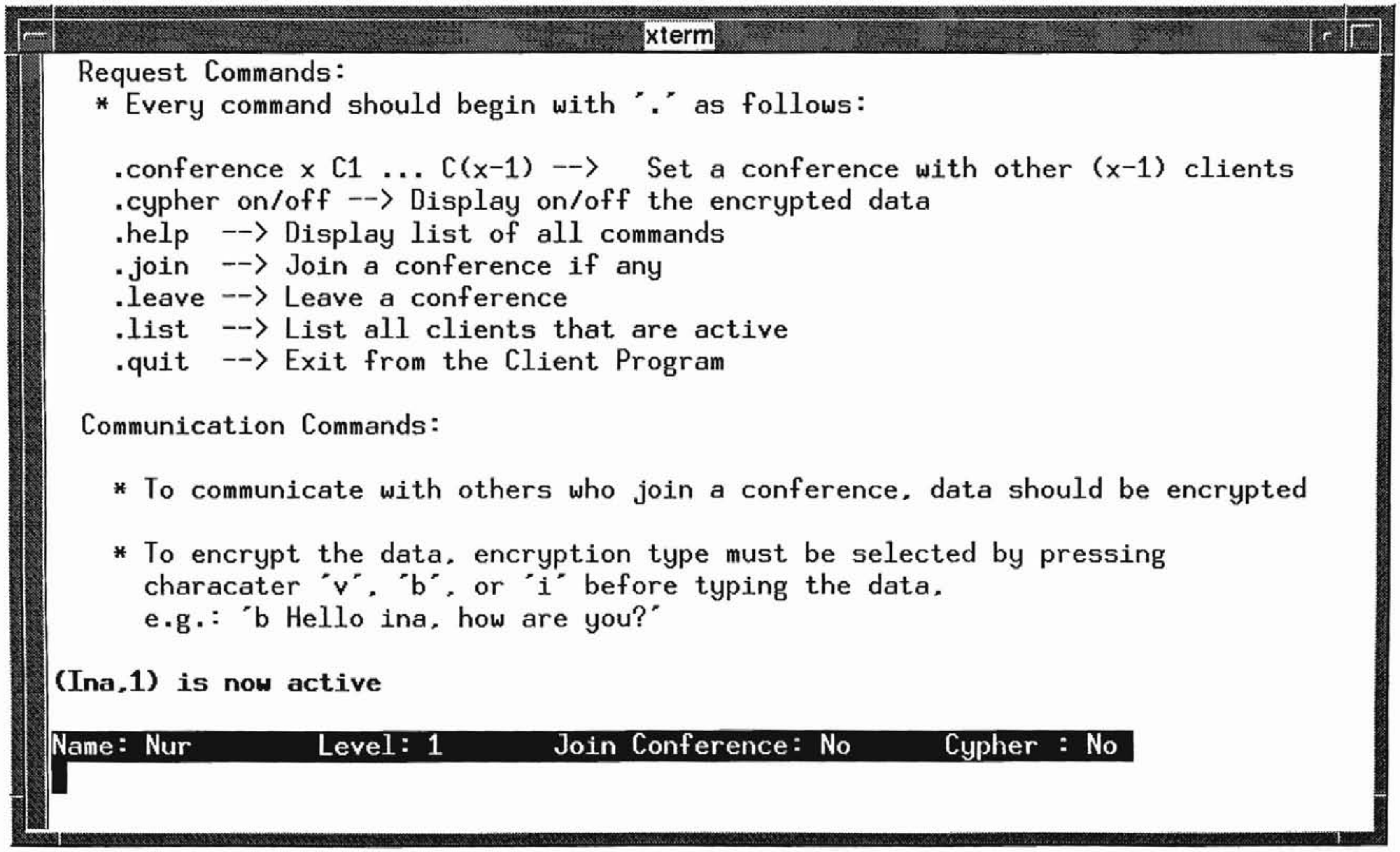
2) *.cypher on/off*. This command is used for displaying cypher data. If the argument *on* is applied, then cypher data will be displayed.
3) *.help*. This command is used for displaying the help menu in the event a client wants to see all of the commands.
4) *.join*. This command is used for joining a conference that is established by other clients.
5) *.leave*. This command is used when a client decides to leave a conference he/she has joined.
6) *.list*. This command is used for listing all client's information.
7) *.quit*. This command is used if a client wants to exit from Secret Conference. In the event that a client has quit, the Server requires another client (process) be created as a substitute of the client who has quit. This should be done because the total number of clients and the composition of the members of the unique subsets should be kept unchanged. Otherwise, the Server will not send all of the information needed to derive the secret communication K.

The second type of command used in Secret Conference is a communication command. This command is used by clients who join a conference to communicate with each other secretly. Secret Conference provides three types of encryption and decryption as described at the beginning of this appendix. Each of the clients should determine the type of encryption and decryption to be used by putting a character 'v' for Vigenere, 'b' for Beaufort, or 'i' for Beaufort Variant, before typing the data to be sent. Suppose, for instance, that client *Nur* wants to communicate with *Ina* who has joined a conference. Then *Nur* may use the Vigenere encryption scheme for encrypting the data to be sent. The format of the interaction could be "v hello Dija how are you?". Client *Ina*, in responding to the message, may use the Beaufort Variant encryption scheme, then the format could be "i hey Nur, I am fine thanks". 'v' and 'i' in these formats indicate the type of encryption and decryption used for communication between *Nur* and *Ina*.



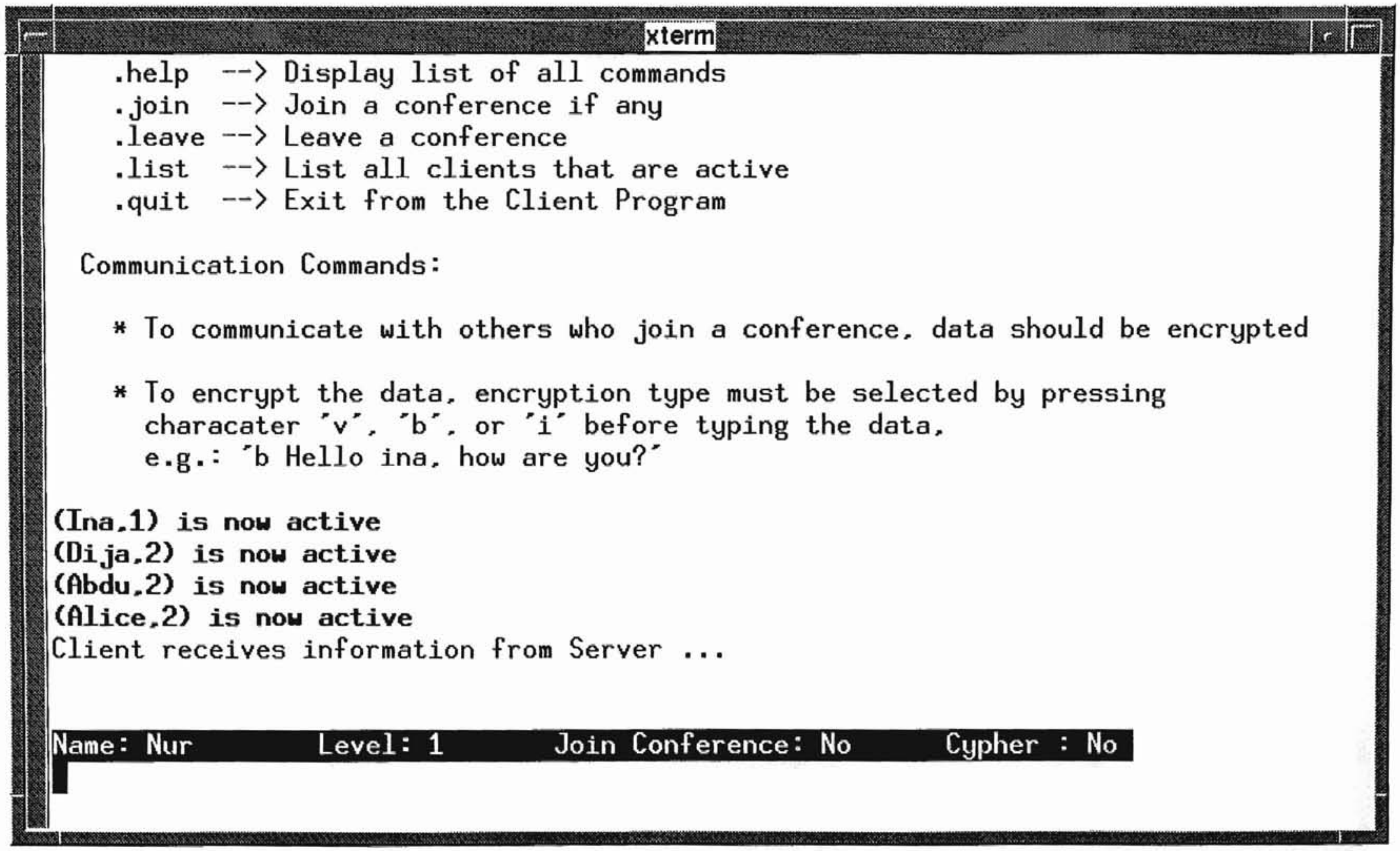
```
Request Commands:
 * Every command should begin with '.' as follows:

  .conference x C1 ... C(x-1) -->   Set a conference with other (x-1) clients
  .cypher on/off --> Display on/off the encrypted data
  .help  --> Display list of all commands
  .join  --> Join a conference if any
  .leave --> Leave a conference
  .list  --> List all clients that are active
  .quit  --> Exit from the Client Program

Communication Commands:

  * To communicate with others who join a conference, data should be encrypted

  * To encrypt the data, encryption type must be selected by pressing
    charactater 'v', 'b', or 'i' before typing the data,
    e.g.: 'b Hello ina, how are you?'

Name: Nur        Level: 1       Join Conference: No       Cypher : No
```

Figure 3. Secret Conference initial screen when the client program is invoked



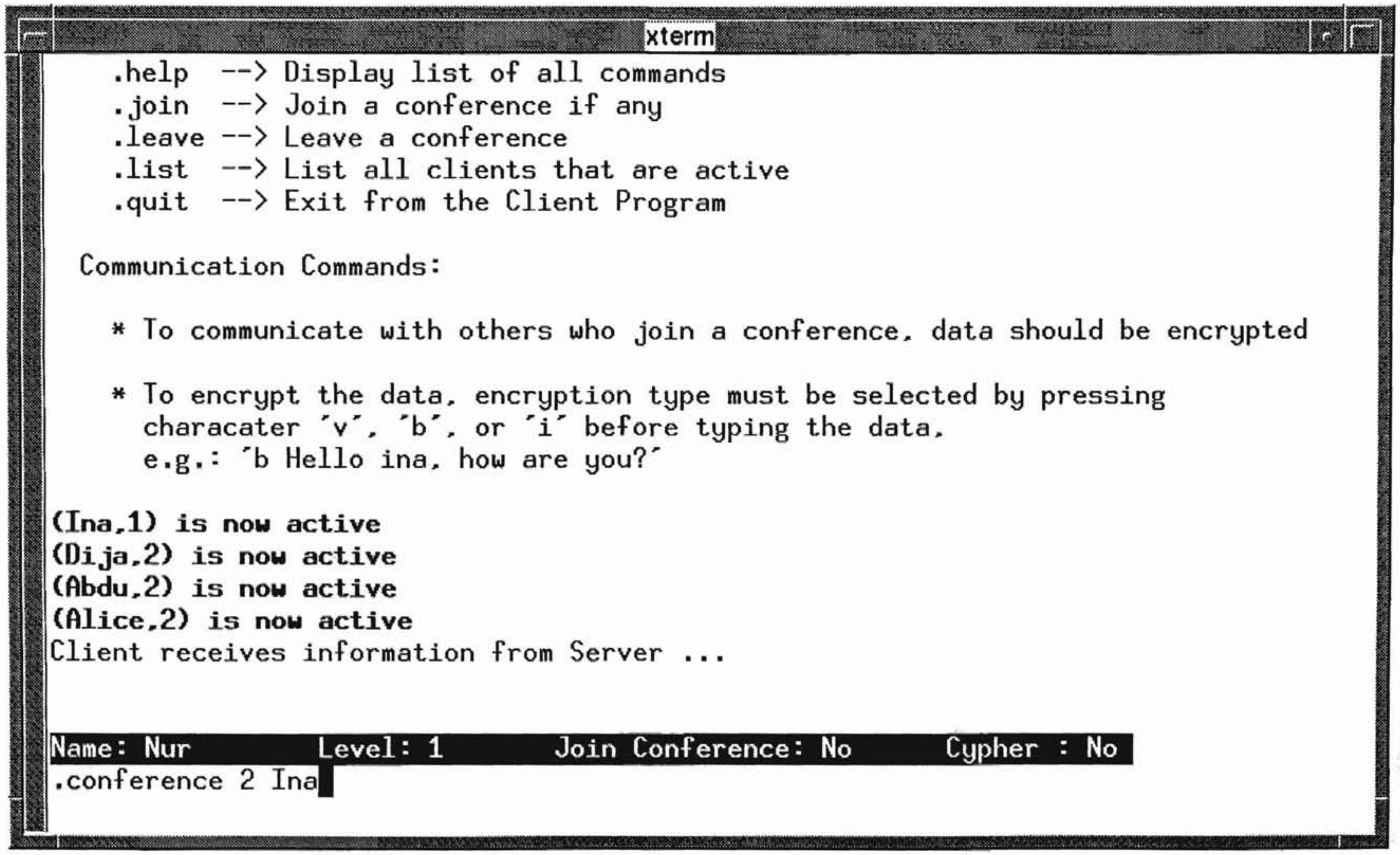
```
Request Commands:
 * Every command should begin with '.' as follows:

   .conference x C1 ... C(x-1) -->   Set a conference with other (x-1) clients
   .cypher on/off --> Display on/off the encrypted data
   .help  --> Display list of all commands
   .join  --> Join a conference if any
   .leave --> Leave a conference
   .list  --> List all clients that are active
   .quit  --> Exit from the Client Program

 Communication Commands:

   * To communicate with others who join a conference, data should be encrypted

   * To encrypt the data, encryption type must be selected by pressing
     characater 'v', 'b', or 'i' before typing the data,
     e.g.: 'b Hello ina, how are you?'

(Ina,1) is now active

Name: Nur          Level: 1        Join Conference: No       Cypher : No
```

Figure 4. Server's message when another client is active



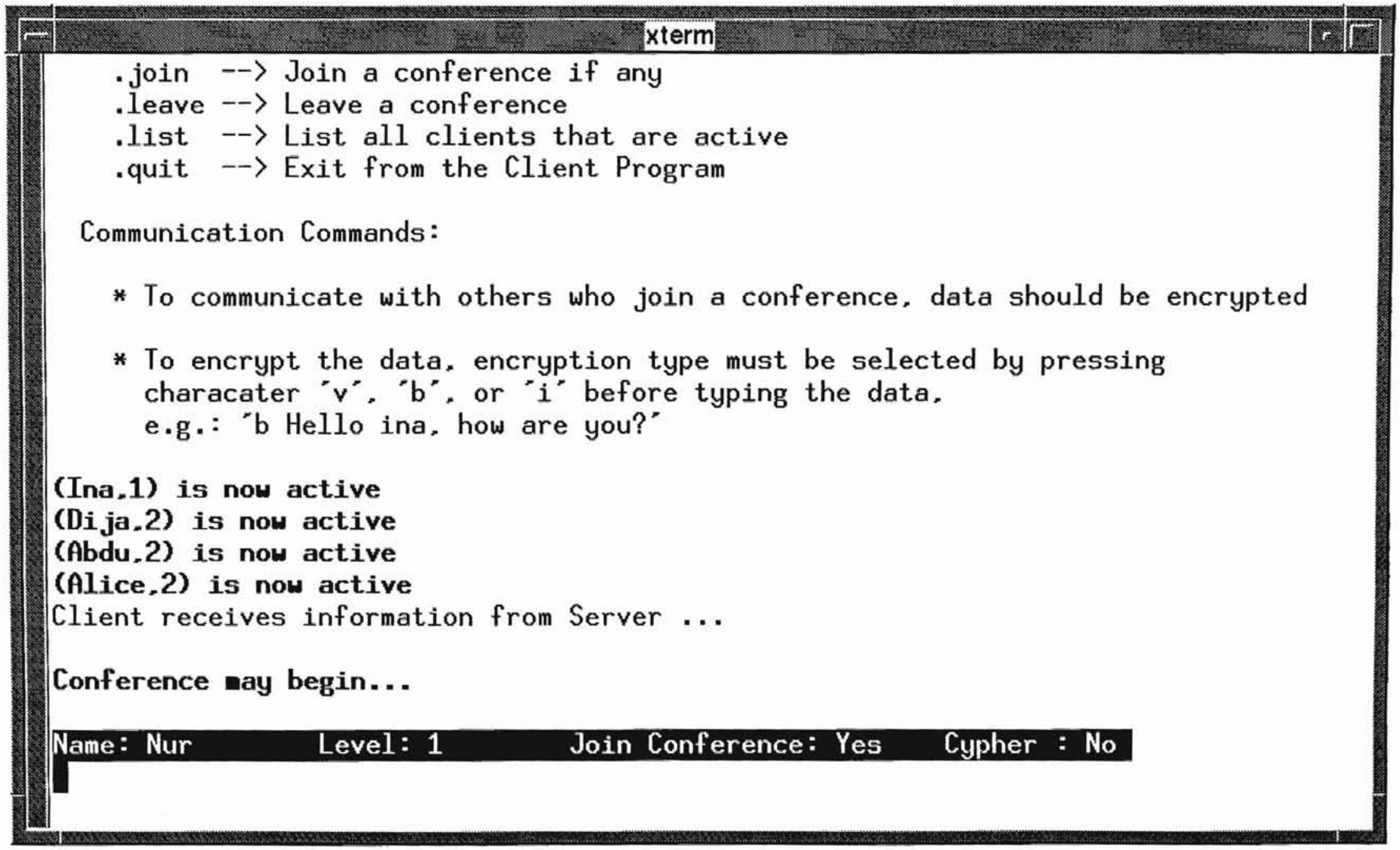
```
    .help  --> Display list of all commands
    .join  --> Join a conference if any
    .leave --> Leave a conference
    .list  --> List all clients that are active
    .quit  --> Exit from the Client Program

  Communication Commands:

    * To communicate with others who join a conference, data should be encrypted

    * To encrypt the data, encryption type must be selected by pressing
      characater 'v', 'b', or 'i' before typing the data,
      e.g.: 'b Hello ina, how are you?'

(Ina,1) is now active
(Dija,2) is now active
(Abdu,2) is now active
(Alice,2) is now active
Client receives information from Server ...


Name: Nur          Level: 1        Join Conference: No       Cypher : No
```

Figure 5. A message displayed when all clients are active



```
xterm
    .help  --> Display list of all commands
    .join  --> Join a conference if any
    .leave --> Leave a conference
    .list  --> List all clients that are active
    .quit  --> Exit from the Client Program

  Communication Commands:

    * To communicate with others who join a conference, data should be encrypted

    * To encrypt the data, encryption type must be selected by pressing
      characater 'v', 'b', or 'i' before typing the data.
      e.g.: 'b Hello ina, how are you?'

(Ina,1) is now active
(Dija,2) is now active
(Abdu,2) is now active
(Alice,2) is now active
Client receives information from Server ...


Name: Nur        Level: 1       Join Conference: No      Cypher : No
.conference 2 Ina
```

Figure 6. Conference request issued by a client



```
    .join  --> Join a conference if any
    .leave --> Leave a conference
    .list  --> List all clients that are active
    .quit  --> Exit from the Client Program

  Communication Commands:

    * To communicate with others who join a conference, data should be encrypted

    * To encrypt the data, encryption type must be selected by pressing
      characater 'v', 'b', or 'i' before typing the data,
      e.g.: 'b Hello ina, how are you?'

(Ina,1) is now active
(Dija,2) is now active
(Abdu,2) is now active
(Alice,2) is now active
Client receives information from Server ...

Conference may begin...

Name: Nur        Level: 1        Join Conference: Yes    Cypher : No
```

Figure 7. A successful reply to a conference request

APPENDIX C

PROGRAM LISTING

The Secret Conference program discussed in Chapter V consists of three files: define.h, Server.c, and Client.c; define.h contains all the header files, constant declarations, and three procedures that are shared by Server.c and Client.c. The Secret Conference program is written in the following order:

```
define.h:
      remaider procedure.
      encrypt procedure.
      decrypt procedure.

Server.c:
      generator procedure.
      matrix operation procedure.
      create_prime procedure.
      power_degree procedure.
      creation_phase procedure.
      create_Z_polynomial procedure.
      divide_PRIME procedure.
      distribution_phase procedure.
      confirm procedure.
      serve_request procedure.
      main procedure.

Client.c:
      obtain_value procedure.
      distribute procedure.
      info_display procedure.
      process_reply procedure.
      info_exchange procedure.
      help procedure.
      main procedure.
```

```
/* ================================================================ */
/* define.h contains all information and procedures that are shared by
   Server.c and Client.c.                                           */
/* ================================================================ */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <float.h>              /* used for matrix operation */
#include <math.h>               /* used in matrix and modular operations */
#include <curses.h>             /* used by Server.c for its definition */
#include <sys/select.h>         /* used in network communication */
```

```
#include <sys/socket.h>                 /* used in network communication */
#include <sys/types.h>                  /* used in network communication */
#include <sys/time.h>                   /* used in network communication */
#include <netinet/in.h>                 /* used in network communication */
#include <arpa/inet.h>                  /* used in network communication */
#include <unistd.h>                     /* used in network communication */

#define UNUSED          -1
#define SERV_HOST_ADDR  "139.78.113.1" /* Server runs on Sequent S/81 */
#define DEFAULT_PORT    66667           /* Port number used by Server */
#define MAXSTRING       256             /* Maximum bytes of data communicated*/
#define BLOCKLENGTH     5               /* Maximum characters per block
                                           encryption/decryption */
#define ALPHABET        94              /* Range of Characters used in
                                           encryption/decryption */

/* ---------------------------------------------------------------------- */
/* Procedure to obtain the result in modular operation (base^power mod prime). */
/* ---------------------------------------------------------------------- */
remainder(int base,int power,int prime)
{
 int result = 1;

  while(power != 0){

    while((power % 2) == 0){            /* while the value of power is even */
      power  /= 2;
      base   *= base;                   /* makes the value of base squared */
      base   %= prime;                  /* get the remainder value of base */
    }

    power   -= 1;                       /* the value of power is odd */
    result  *= base;
    result  %= prime;                   /* get the remainder value of result */
  }
  if(result < 0) result *= -1;
  return result;
}

/* ---------------------------------------------------------------------- */
   /*  Procedure  to  encrypt  data  to  be  sent;  char.  option  used  for
    determining encryption type: v for Vigenere, b for Beaufort, and i for
    Beaufort Variant; char. *key used as the secret key; char. *plain used
    for  plain  text;  char.  *cypher  as  cypher  text;  and  int.  for_confr.
    indicates encryption used for conference or not. */
/* ---------------------------------------------------------------------- */
encrypt(char option,char *key,char *plain,char *cypher,int for_confr)
{
 int i,j;
 int string_length,key_length;          /* used for obtaining the lengths of
                                           the secret key and the plain data */
 int a,c;
 char temp[MAXSTRING];                  /* temporary buffer */

  a             = ' ';
  option        = toupper(option);      /* get the type of encryption */
  key_length    = strlen(key);          /* length of the secret key K */
  string_length = strlen(plain);        /* length of data to be enctypted */
  i = j = 0;

  /* Encrypt the data according to type of encryption chosen */
  while(i < string_length){
   if((char)option == 'V')              /* if encryption chosen is Vigenere */
     c = ((int)plain[i] + (int)key[i%key_length] - (2 * a)) % ALPHABET;
   else if((char)option == 'B')         /* Beaufort encryption scheme */
     c = ((int)key[i%key_length] - (int)plain[i]) % ALPHABET;
   else c = ((int)plain[i] - (int)key[i%key_length]) % ALPHABET;
   if(c <= 0) c += ALPHABET;
```

```
    temp[j] = (char)(c + a);
    j += 1; i += 1;
    if((i % BLOCKLENGTH) == 0){        /* group the encrypted data per block */
      temp[j] = ' ';
      j       += 1;
    }
  }

  temp[j]   = '\n';
  temp[j+1] = '\0';
  /* determine whether the data is request command or communication command */
  if(for_confr == TRUE) sprintf(cypher,"%c %s",option,temp);
  else strcpy(cypher,temp);
}

/* ------------------------------------------------------------------------ */
    /* Procedure to decrypt data to be sent; char. option used for
   determining encryption type: v for Vigenere, b for Beaufort, and I for
   Beaufort Variant; char. *key used as the secret key; char. *plain used
   for plain text; char. *cypher as cypher text; and int. for_confr.
   indicates decryption used for conference or not. */
/* ------------------------------------------------------------------------ */
decrypt(char option,char *key,char *plain,char *cypher,int for_confr)
{
 int i,j,k;
 int a,c;
 int key_length;        /* used for obtaining the length of the secret */
 char temp[6];          /* temporary buffer */

  a          = ' ';
  option     = toupper(option);        /* get the encryption type */
  key_length = strlen(key);
  j          = 0;
  if(for_confr == TRUE){
   strtok(cypher," ");
   strcpy(cypher,strtok('\0',"\n"));
  }

  while(sscanf(cypher,"%s",temp) != EOF && strcmp(temp,NULL) !=0){
   i  = 0;
   k  = strlen(temp);

   /* decrypt the data according to the type of encryption chosen */
   while(i < k){
     if((char)option == 'V')           /* if encryption chosen is Vigenere */
       c = ((int)temp[i] - (int)key[j%key_length]) % ALPHABET;
     else if((char)option == 'B')      /* encryption chosen is Beaufort */
       c = ((int)key[j%key_length] - (int)temp[i]) % ALPHABET;
     else c = ((int)temp[i] + (int)key[j%key_length] - (2 * a)) % ALPHABET;
     if(c < 0) c += ALPHABET;
     plain[j] = (char)(c + a);
     j += 1; i += 1;
   }
   strtok(cypher," ");
   strcpy(cypher,strtok('\0',"\n"));
  }
  plain[j]   = '\n';
  plain[j+1] = '\0';
}

/* ================================================================= */
/* Server.c contains elevent  procedures used for simulating a server in
   Secret Conference.                                                  */
/* ================================================================= */
#include "define.h"

#define YES       10
#define LIMIT1    10000 /* used for generating coeffs' values
```

```
                                of polynomials */
#define LIMIT2      1000    /* used for creating shadows, prime, etc. */
#define LIMIT3      50      /* used for x part's value of polynomials */

int      *shadow;           /* Shadows of The secret key */
int      *Prime;            /* used for obtaining a communication key used
                               Server & a client */
int      *z1_x;             /* used for x parts of Z1 polynomial */
int      *z2_x;             /* used for x parts of Z2 polynomial */
int      *t_x;              /* used for x parts of other polynomial */

int      PRIME;             /* used for obtaining the coefficients of Y(x) */
int      degree;            /* degree of Y(x) */
int      conference;        /* used for determining whether a conference
                               exists or not */
int      Start;             /* used for indicating whether distribution_phase
                               works or not */
int      Client;            /* used for obtaining the active active number */

double **S;                 /* Shadow Matrix */
double **Inv;               /* Inverse Matrix of Shadow Matrix */
double *C;                  /* Matrix of cofficients of Y(x) */
double *z1_coef;            /* coefficients of Z1 polynomial */
double *z2_coef;            /* coefficients of Z2 polynomial */
double *t_coef;             /* coefficients of other polynomial */
double *Z1;                 /* values of Z1 polynomial */
double *Z2;                 /* values of Z2 polynomial */
double *T;                  /* values of other polynomial */

struct client{
  int    sockfd;            /* socket description used as a port for communication
                               with a client */
  char   id[10];            /* client's name  */
  char   key[12];           /* communication key used by Server & a client */
  int    shadow;            /* shadow */
  int    level;             /* group to which a client belongs */
  int    index;             /* index for obtaining shadow */
  int    joint;             /* indicating a client join a conference or not */
  int    given;             /* indicating a client is given the
                               distributed  information or not */
  struct client *next;
};

struct client *root;

/* ------------------------------------------------------------------------ */
   /* Procedure to generate a random value; int. prime indicates that the
   random value generated is prime or not. */
/* ------------------------------------------------------------------------ */
generator(int prime)
{
 int i = 1;
 int value,temp = 0;

  while((value = rand()) == 0);          /* get some random value */
  if(prime == TRUE){                     /* if the value intended to be prime */
    do{
      if(i <= temp) value = rand();
        temp = ceil(sqrt(value));        /* get the square root of the value */
        for(i = 2;i <= temp;i++)
          if((value % i) == 0) break;    /* the value generated is not prime */
    }while(i <= temp);
  }
  return value;
}
```

```
/* ------------------------------------------------------------------------- */
  /* Procedure to find the inverse matrix of shadow matrix shown in
  Equation (3.5); int. dim is the degree of Y(x);int.first_time indicates
  whether the procedure is called for the first time or not;int. n is the
  total client number. */
/* ------------------------------------------------------------------------- */
matrix_operation(int dim,int first_time,int n)
{
 int    i,j,k,l,error;
 int    row,col;                 /* row & column variables */
 int    result;                  /* identify the result good or not */
 double data;                    /* temporary data */
 double data1,data2;             /* temporary data */
 double *temp1,*temp2;           /* temporary matrices */

  /* Create temporary buffer as temporary matrices */
  temp1 = calloc(dim,sizeof(double));
  temp2 = calloc(dim,sizeof(double));
  if(temp1 == NULL || temp2 == NULL){
   printf("Fatal Error in calloc temp\n");
   exit(1);
  }
  /* If the procedure is called for the first time */
  if(first_time == TRUE){
    /* Create all matrices needed for obtaining the inverse matrix */
   shadow= calloc(n,sizeof(int));
   C      = calloc(dim,sizeof(double));
   S      = calloc(dim,sizeof(double *));
   Inv    = calloc(dim,sizeof(long double *));
   if(shadow == NULL || C == NULL || S == NULL || Inv == NULL){
     printf("Fatal Error in calloc shadow etc.\n");
     exit(1);
   }
   for(i = 0;i < dim;i++){
     S[i]   = calloc(dim,sizeof(double));
     Inv[i] = calloc(dim,sizeof(long double));
     if(S[i] == NULL || Inv[i] == NULL){
       printf("Fatal Error in calloc S[i] or Inv[i]\n");
       exit(1);
     }
   }
   for(i = 0;i < dim;i++) Inv[i][i] = 1; /* Initialize inverse matrix as I */
  }else{      /* The procedure is called for the second time or more */
    /* Initialize matrices used for the operations */
   for(i = 0;i < dim;i++){
     for(j = 0;j < dim;j++){
       S[i][j] = Inv[i][j]  = 0;
       if(i == j) Inv[i][j] = 1;
     }
   }
  }

  i = 0;

  /* Put the values [(s)mod m], ...,[(s^r)mod m] into matrix S */
  while(i < dim){
   if(first_time == FALSE && i < n) k = shadow[i];  /* get a shadow from S */
   else k = (generator(FALSE) % LIMIT2);            /* create a new shadow */
   l = 0;
   while(l < i && k != 0) if(S[l++][0] == k) break;
   if(l == i && k != 0){
     for(j = 1;j <= dim;j++) S[i][j-1] = remainder(k,j,PRIME);
     if(first_time == TRUE && i < n) shadow[i] = S[i++][0];
   }
  }
 /* Start finding the inverse matrix using the second method of finding an
 inverse matrix described in Section 2.1 */
error = FALSE;
for(i = 0;i < dim;i++){
   /* Applying the first step of the method */
  if(S[i][i] == 0){       /* If the entries (i,i) is zero */
    col = i; row = i + 1;

    while(row < dim) if(S[row++][col] != 0) break;
    if(row < dim){         /* Swap all the entries */
      memcpy(temp1,S[row],dim);
      memcpy(temp2,Inv[row],dim);
      memcpy(S[row],S[i],dim);
      memcpy(Inv[row],Inv[i],dim);
      memcpy(S[i],temp1,dim);
      memcpy(Inv[i],temp2,dim);
    }else{
      error = TRUE;
      printf("in error\n");
    }
  }

  if(error == FALSE){
    row = col = i;

    /* Applying the second step of the method */
    while(row < dim){
      if(S[row][col] != 1){
        data = S[row][col]; k = 0;
        while(k < dim){
          S[row][k]    = S[row][k] / data;
          Inv[row][k]  = Inv[row][k] / data;
          k           += 1;
        }
      }
      row += 1;
    }

    /* Applying the thirs step of the method */
    j = 0; row = col = I;

    while(j < dim){
      if(j != row && S[j][col] != 0){
        data = S[j][col]; k = 0;

        /* Subtract all entries in row i from the corresponding entries
           in row j */
        while(k < dim){
          data1       = (data * S[row][k]);
          data2       = (data * Inv[row][k]);
          S[j][k]    -= (fmod(data1,PRIME));
          Inv[j][k]  -= (fmod(data2,PRIME));
          S[j][k]     = (fmod(S[j][k],PRIME));
          k          += 1;
        }
      }
      j += 1;
    }
  }
 /* Repeat all steps starting from step 1 */
 }
 free(temp1); free(temp2);
 return error;
}
```

```
/* ------------------------------------------------------------------------ */
   /* Procedure to create prime numbers; int. n is the total client number;
     int all indicates if all primes should be created. */
/* ------------------------------------------------------------------------ */
void create_primes(int n,int all)
{
 int i,j;    /* Temporary variables */

  if(all == TRUE){          /* If all primes needed to be created */
    /* Create buffer for storing the first n primes */
   if((Prime = calloc(n,sizeof(int))) == NULL){
     printf("Fatal Error in calloc Prime\n");
     exit(1);
   }

   /* Create the first n primes */
   i = 0;

   while(i < n){
     Prime[i] = generator(TRUE);
     for(j = i-1;j >= 0;j--) if(Prime[i] == Prime[j]) break;
     if(j < 0) i += 1;
   }
  }

  /* Next, create the prime used for modular computation */
  do{
   PRIME = generator(TRUE);
   for(i = 0;i < n;i++) if(PRIME == Prime[i] || PRIME < LIMIT2) break;
  }while(i < n);
}

/* ------------------------------------------------------------------------ */
   /*Procedure to obtain the degree of polynomial Y(x); int. n is the total
     client number; int. u1 is the size of the first unique subset. */
/* ------------------------------------------------------------------------ */
power_degree(int n,int u1)
{
  return (u1 + 1)*(n - u1);
}

/* ------------------------------------------------------------------------ */
   /* Procedure to create all information needed by clients to obtain the
     secret key K used for conference; int. u1 is the size of the first
     unique subset; int. first_time indicates if the procedure is called
     for the first time. */
/* ------------------------------------------------------------------------ */
creation_phase(int u1,int n,int first_time)
{
 int   i,j;               /* Temporary variables */
 int   error;             /* Indicates an error occured */
 int   secret_key;        /* Used for storing the secret key K */
 double data,temp;        /* Temporary variables */

  if(first_time == TRUE){ /* If the procedure is called for the first time */
   create_primes(n,TRUE); /* Create all needed primes */
   degree = power_degree(n,u1);             /* Obtain the degree of Y(x) */
   error  = matrix_operation(degree,TRUE,n); /* Get the inverse matrix */
  }else{                  /* It is called for the second time or more */
   create_primes(n,FALSE);
   error  = matrix_operation(degree,FALSE,n);
  }

  while(error == TRUE) error = matrix_operation(degree,FALSE,n);

  while((secret_key = (generator(FALSE)*101)) == 0);  /* Create the secret K */
  /* Obtain the coefficients of Y(x) as shown in Equation (3.7) */
  for(i = 0;i < degree;i++){
    j = 0; data = 0;

    while(j < degree) data  += Inv[i][j++];
    data *= secret_key; C[i] = data;
  }
  return secret_key;
}

/* ---------------------------------------------------------------------- */
   /* Procedure to create polynomials of Z(x)s and other if any; int.
     for_u2 indicates whether or not the polynomial(s) for the second
     unique subset;  double *coef is used for the coefficients of the
     polynomial; int *x is used for the x parts of the polynomial. */
/* ---------------------------------------------------------------------- */
void create_Z_polynomial(int for_u2,double *coef,int *x)
{
 int    power;              /* Indicates the degree of Z(x) polynomial */
 int    total_x;            /* Indicates the number of x parts of Z(x) */
 int    i,j;                /* Temporary variables */
 double value;              /* Indicates a random value */

  power    = coef[0];
  total_x  = x[0];
  for(i = 1;i <= power;i++){

    /* Get a random value from random generator */
    while((value = ((generator(FALSE) % LIMIT1) / 100)) == 0 );
    /* Put the value as a coefficient of Z(x) */
    coef[i] = pow(-1,(generator(FALSE) % 2)) * value;
  }
  /* Put x parts of the polynomial as negative values */
  if(for_u2 == UNUSED) for(i = 0;i < total_x;i++) x[i] = (-1) * (i + 1);
  else{
    for(i = 0;i < total_x;i++){

      while((value = generator(FALSE) % LIMIT3) == 0);
      j = 0;

      /* If some values of x genereated by the generator are same */
      while(j < i) if(x[j++] == value) break;
      if(j < i) i -= 1;

      else x[i]    = value;
    }
  }
}

/* ---------------------------------------------------------------------- */
   /* Procedure to divide PRIME (m) into I_ms; int equation indicates which
   case occurs; int. create indicates whether or not Z(x) and other
   polynomials should be created; int. u1 is the size of the first uniqur
   subset; int. u2 is that of the second one; int. w is the weight of the
   first unique subset's members over the second's. */
/* ---------------------------------------------------------------------- */
void divide_PRIME(int equation,int create,int u1,int u2,int w)
{
 int    i,j;                      /* Temporary variables */
 double value1,value2;            /* Temporary values of Z(x)s for U1 and U2 */

  if(create == TRUE){             /* If Z(x) should be created */
    z2_coef[0] = u2 - 1;          /* The degree of Z(x) for U2 */
    z2_x[0]    = w * u1 + u2;     /* Number of x's values should be created */
    create_Z_polynomial(TRUE,z2_coef,z2_x);
    if(equation != 2){            /* If case I or III of dividing m occurs */
      z1_coef[0] = u1 - 1;        /* The degree of Z(x) for U1 */
      z1_x[0]     = u1;            /* Number of x's values should be created */
      create_Z_polynomial(FALSE,z1_coef,z1_x);
    }
  }

  /* Divide/Partition m into Ims */
  for(i = 0;i < w * u1 + u2;i++){
    value1 = PRIME;
    if(equation != 2 && i < u1) value2 = PRIME;
    for(j = 1;j < u2;j++){
      value1 += z2_coef[j] * pow(z2_x[i],j);
      if(equation != 2 && j < u1) value2 += z1_coef[j] * pow(z1_x[i],j);
    }
    Z2[i] = value1;
    if(equation != 2 && i < u1) Z1[i] = value2;
  }
  if(equation == 3){               /* If Case III of dividing m occurs */
    for(i = 0;i < w;i++){
      for(j = 1;j < u1;j++){       /* Set same Im of Z2(x) for U1 */
        z2_x[i+j*w] = z2_x[i];
        Z2[i+j*w]   = Z2[i];
      }
    }
  }
}

/* ------------------------------------------------------------------------ */
   /* Procedure to distribute all the required information needed by client
      to obtain the secret  used for secret conference;int. first_time
      indicates whether the procedure is called for the first time; int.
      create_coef indicates whether Z(x)s and other polynomials should be
      created; int.n is the total clients; int u1 is the size of the first
      unique subset; int. w is the weight.*/
/* ------------------------------------------------------------------------ */
distribution_phase(int first_time,int create_coef,int n,int u1,int w)
{
 int      i,j,k;                /* Temporary variables */
 int      index;                /* Used as index of x's values and of Z(x)'s
                                   values */
 int      equation;             /* Used to indicate which case occurs(Case I, II
                                   or III) (see Section 3.2.2) */
 int      u2,u3;                /* Indicates sizes of subsets U1 and U2 */
 int      choice;               /* Indicates type of condition of distributing
                                   the coeffs. of Y(x) for U2 */

 double   value;                /* Temporary values for θ(x) */

 char     temp[20];             /* Temporary buffer */
 char     temp1[MAXSTRING];     /* Temporary buffer for sending/receiving
                                   data to be communicated */
 char     temp2[MAXSTRING];     /* Temporary buffer for sending/receiving
                                   data to be communicated */
 char     buffer[MAXSTRING];    /* Temporary buffer for sending/receiving
                                   data to be communicated */

 struct client *cur;

  u2 = n - u1;                  /* Size of U2 */
  if(first_time == TRUE){       /* If the procedure is called the first time */
   /* Create buffer for all information related to Z(x) */
   z2_coef = calloc(u2,sizeof(double));
   z2_x    = calloc(u1 * w + u2,sizeof(int));
   Z2      = calloc(u1 * w + u2,sizeof(double));
   if(z2_coef == NULL || z2_x == NULL || Z2 == NULL){
     printf("Fatal Error in Z2 family\n");
     exit(1);
   }
 if(u1 * w < u2 || (u1 -1) * w >= u2){
   /* Create another Z(x) if Case I or III of partitioning m occurs */
   z1_coef  = calloc(u1,sizeof(double));
   z1_x     = calloc(u1,sizeof(int));
   Z1       = calloc(u1,sizeof(double));
   if(z1_coef == NULL || z1_x == NULL || Z1 == NULL){
     printf("Fatal Error in Z1 family\n");
     exit(1);
   }
  }
}
/* Determine which Case of partitioning m occurs */
if(u1 * w < u2) equation = 1;  /* Case I occurs */
else if(u1 * w >= u2 && (u1 - 1) * w < u2) equation = 2;/* Case II occurs */
else equation = 3;               /* Case III occurs */

divide_PRIME(equation,create_coef,u1,u2,w);

/* Determine which condition of distr. the coeffs. of Y(x) for U2 occurs */
if(equation != 3){
  if((u1 -1) * w + 1 >= u2)       choice = 1; /* One member of U2 in the non-
                                                 unique subsets */
  else if((u1 - 1) * w + 2 >= u2) choice = 2; /* Two members of U2 in the
                                                 non-unique subsets */
  else choice = 3;                            /* Three members or more of U2
                                                 in the non-unique subsets */
}else{
  if(w + 1 >= u2)      choice = 1;
  else if(w + 2 >= u2) choice = 2;
  else choice = 3;
}
/* Create buffer for storing all info. related to θ(x)
   if choice = 3 occurs */
if(choice == 3 && first_time == TRUE){
  if(equation != 3) t_coef = calloc(u2 - (u1 - 1) * w,sizeof(double));
  else t_coef= calloc(u2 - 1,sizeof(double));
  t_x          = calloc(u2,sizeof(int));
  T            = calloc(u2,sizeof(double));
}

cur = root;

/* Start distributing all needed info. to clients */
while(cur != NULL){
  if(cur->sockfd != UNUSED && cur->given == FALSE){
    /* If the client is a member of U1 */
    if(cur->level == 1){
      /* Put Imi into buffer */
      if(equation == 1 || equation == 3)
        sprintf(temp1,": (%d,%f) ",z1_x[cur->index],Z1[cur->index]);
     for(i = 0;i < w;i++){
        sprintf(temp,"(%d,%f) ",z2_x[cur->index*w+i],Z2[cur->index*w+i]);
        strcat(temp1,temp);
      }
      /* Append some coeffs. of Y(x) to the buffer */
      strcat(temp1," ; ");
      for(i = 0;i < degree;i++){
        if(i == (cur->index * u2) && u1 > 1){
          i += u2 -1;
          if(choice == 3){
            sprintf(temp,"(%d,%d,%f) ",-1,i+1,0);
            strcat(temp1,temp);
          }
        }else{
          sprintf(temp,"(%d,%f) ",i+1,C[i]);
          strcat(temp1,temp);
        }
  }
/* The client is a member of of U₂ */
}else{
  /* Put I_mi into buffer */
  sprintf(temp1,"(%d,%f)",z2_x[(w-1)*u1+cur->index],
          Z2[(w-1)*u1+cur->index]);
  strcat(temp1,";");
  /* Append some coeffs. of Y(x) to the buffer
     according to the value of choice */
  for(i = 0;i < degree;i++){
    if(i < degree - u2){      /* For the first |U₁|*|U₂| coeffs. of Y(x) */
      if(choice == 1){        /* Give all of the coeffs. to the member */
        sprintf(temp,"(%d,%f) ",i+1,C[i]);
        strcat(temp1,temp);
      }else if(choice == 2){ /* See Section 3.2.2 condition 2 */
        if((i % u2) != (cur->index - u1)){
          sprintf(temp,"(%d,%f) ",i+1,C[i]);
          strcat(temp1,temp);
        }
      }else{                  /* See Section 3.2.2 condition 3 */
        if((i % u2) != (u2 - 1)){
          if((u2 - 1) == 3){
            if((i % u2) != (cur->index - u1) && cur->index != u2 - 1){
              sprintf(temp,"(%d,%f) ",i+1,C[i]);
              strcat(temp1,temp);
            }
          }else{
            j = i % u2; u3 = u2 - 1; index = cur->index;
            if(j!=index % u3 && j != (index + 1) % u3 && index != u3){
              sprintf(temp,"(%d,%f) ",i+1,C[i]);
              strcat(temp1,temp);
            }
          }
        }else{
          if(cur == root){
            if(equation == 3) t_coef[0] = u2 - 1;
            else t_coef[0] = u2 - (u1 - 1) * w - 1;
            t_x[0] = u2;
            create_Z_polynomial(UNUSED,t_coef,t_x);
            for(j = 0;j < u2;j++){
              value = C[i];
                for(k = 1;k < t_coef[0];k++)
                  value += t_coef[k] * pow(t_x[j],k);
              T[j]  = value;
            }
          }
          sprintf(temp,"(%d,%d,%f) ",t_x[cur->index-u1],i+1,
                  T[cur->index - u1]);
          strcat(temp1,temp);
        }
      }
    }else{      /* For the last |U₂| coeffs. of Y(x) */
      if(((i % u2) != (cur->index - u1))){      /* Give one coeff. for
                                                   each member of U₂ */
        sprintf(temp,"(%d,%f) ",i+1,C[i]);
        strcat(temp1,temp);
      }
    }
  }
}
/* Send the information to the clients */
do{
  sprintf(buffer,".distribute %d %s",shadow[cur->index],temp1);
  encrypt('v',cur->key,buffer,temp2,FALSE);
  writen(cur->sockfd,temp2,strlen(temp2));
  readline(cur->sockfd,buffer,MAXSTRING);
  decrypt('v',cur->key,temp2,buffer,FALSE);
```

```
    }while(strncmp(temp2,"OK",2) != 0);
    cur->given = TRUE;
   }
   cur = cur->next;
  }
 return FALSE;
}

/* ------------------------------------------------------------------ */
  /* Procedure to exchange information between Server and a client; int.
    sockfd is the client's socket used for communication; int. n is the
    total clients; int. u1 is the size of the first unique subset; int. w
    is the weight. */
/* ------------------------------------------------------------------ */
info_exchange(int sockfd,int n,int u1,int w)
{
 char data[20];           /* Temporary buffer for client ID */
 char key[15];            /* Used as a private secret comm. key between Server
                             and a client */
 char buffer[MAXSTRING]; /* Temporary buffer communication */
 char buf[MAXSTRING];    /* Temporary buffer communication */
 char buff[MAXSTRING];   /* Temporary buffer communication */
 int  i,j,k;
 int  a;                  /* Used as base in a^private mod prime */
 int  private;            /* Used as a power in a^private mod prime */
 int  prime;              /* Used as a prime in a^private mod prime */

 struct client *prev,*cur,*temp;

  while((a = (generator(FALSE)%LIMIT3)) == 0);/* Get some value used as base */

  prime = Prime[rand()%n];
  while((private = generator(TRUE)) > (LIMIT2/10));

  while((j = remainder(a,private,prime)) == 0);
  /* Put a, prime, and j = a^private mod prime into buffer & send them to client */
  sprintf(buffer,"%d %d %d\n\0",a,prime,j);
  writen(sockfd,buffer,strlen(buffer));

  /* Wait for response from the client & get the private key */
  readline(sockfd,buffer,MAXSTRING);
  i = remainder(atoi(buffer),private,prime);
  sprintf(key,"%d",i);

  /* Wait response from the client for checking the private key */
  readline(sockfd,buffer,MAXSTRING);
  memset(buff,'\0',strlen(buff));
  decrypt('v',key,buff,buffer,FALSE);

  /* If the data sent by the client is true, info. exchange begins */
  if((i = atoi(buff)) == j){
    /* Send the confirmation to the client */
    sprintf(buff,"OK: send in the data in B");
    encrypt('v',key,buff,buffer,FALSE);
    writen(sockfd,buffer,strlen(buffer));

    /* Wait for response from the client to get all info. about the client */
    readline(sockfd,buffer,strlen(buffer));
    decrypt('b',key,buff,buffer,FALSE);
    sscanf(buff,"%s %d",data,&j);
    sprintf(buffer,"%d %d:",degree,w);

    /* Send the info. of the other clients who have established connections to
       this new client */
    cur = root;
    while(cur != NULL){
      if(cur->sockfd != UNUSED){
        if(strcmp(cur->id,data) != 0){
```

```
      /* Put the other clients' info. into buffer */
      sprintf(buf,"(%s,%d) ",cur->id,cur->level);
      strcat(buffer,buf);
    }
  }
  cur = cur->next;
}
strcpy(buf,buffer);

do{
  /* Encrypt the info. using Vigenere scheme & send it to the client */
  encrypt('v',key,buf,buffer,FALSE);
  writen(sockfd,buffer,strlen(buffer));

  /* Wait for response from the client */
  readline(sockfd,buffer,MAXSTRING);
  decrypt('v',key,buff,buffer,FALSE);
}while(strncmp(buff,"OK",2) != 0);

prev = NULL; cur = root; i = UNUSED;
k = 0;

/* Allocate free structure for storing the new client's info. */
while(cur != NULL){
  k += 1;
  if(cur->sockfd == UNUSED && cur->level == j){
    i = cur->index;
    break;
  }
  if(cur->level == j) i = cur->index + 1;
  prev = cur; cur = cur->next;
}
if(cur == NULL)temp = (struct client *) malloc(sizeof(struct client));
else if(cur != NULL) temp = cur;
else if(k > n){
  printf("ERROR IN info_exchange(i.e. # of struct clients > %d\n",n);
  exit(1);
}

/* Put the new client's info. into the free structure */
temp->sockfd = sockfd;
temp->level  = j;
strcpy(temp->id,data);
strcpy(temp->key,key);
temp->joint  = temp->given = FALSE;
if(i == UNUSED){
  if(j == 1) temp->index      = 0;
  else temp->index   = u1;
}else temp->index  = i;
temp->shadow = shadow[temp->index];
if(cur == NULL) temp->next   = NULL;
else temp->next = cur->next;
if(prev == NULL) root = temp;
else prev->next = temp;

/* Send the other client who have established the connections that the new
   client is active */
cur = root;
sprintf(buffer,".participate %s %d",temp->id,temp->level);
sprintf(buf,"(%s,%d) is now active",temp->id,temp->level);
while(cur != NULL){
  if(cur->sockfd != UNUSED && cur != temp){
    encrypt('v',cur->key,buffer,buff,FALSE);
    writen(cur->sockfd,buff,strlen(buff));
    encrypt('v',cur->key,buf,buff,FALSE);
    writen(cur->sockfd,buff,strlen(buff));
  }
  cur = cur->next;
```

```
    }
  /* The response from about the private key is not valid */
  }else{
    /* Server send negative response to the client */
    sprintf(buff,"Server receives invalid data !!!");
    encrypt('v',key,buff,buffer,FALSE);
    writen(sockfd,buffer,strlen(buffer));
    close(sockfd);
    j = FALSE;
  }
  return j;
}

/* ----------------------------------------------------------------------- */
   /* Procedure to check whether or not the subset of clients who want to
   make a conference is a qualified subset; if it is, the request is
   processed by server_request procedure; if it is not, the request is
   rejected; int. number is the total clients in the subset; char. **id
   contains the name of the clients; int. n is the total clients;int u1 is
   the size of the first unique subset; int. w is the weight. */
/* ----------------------------------------------------------------------- */
confirm(int number,char **id,int n,int u1,int w)
{
 int i;               /* Temporary variable */
 int u2;              /* Size of U2 */
 int decision;        /* Used for granting or rejecting the request */
 int level1,level2;   /* Indicates levels of the clients (i.e. level 1 or 2) */
 int equation;        /* Indicates Case of partitioning m into Ims */

 struct client *cur;

  u2 = n - u1;        /* Get size of U2 */

  /* Determine which Case of partitioning m occurs */
  if(u1 * w < u2) equation = 1;
  else if(u1 * w >= u2 && (u1 - 1) * w < u2) equation = 2;
  else equation = 3;

  /* Count member's level of clients who issue a conference request */
  level1 = level2 = 0;
  cur = root;
  while(cur != NULL){
    if(cur->sockfd != UNUSED){
      for(i = 0;i < number;i++){
        if(strcmp(id[i],cur->id) == 0){
          if(cur->level == 1) level1 += 1;
          else level2 += 1;
          break;
        }
      }
    }
    cur = cur->next;
  }

  if(level2 == 0){                      /* The clients are from U1 */
    if(level1 == u1) decision = TRUE;   /* All of the clients are from U1 */
    else decision = FALSE;              /* The clients are only a subset of U1 */
  }else if(level1 == 0){                /* The clients are from U2 */
    if(level2 == u2) decision = TRUE;   /* All of the clients are from U2 */
    else decision = FALSE;              /* The clients are only a subset of U1 */
  }else if(equation == 3){              /* Case III of dividing m occurs */
    if(w + level2 >= u2) decision = TRUE;
    else decision = FALSE;
  }else{                                /* Case I or II occurs */
    if(level1 * w + level2 >= u2) decision = TRUE;
    else decision = FALSE;
  }
 return decision;
}

/* ------------------------------------------------------------------------ */
   /* Procedure to serve a request sent by a client; there are several
   services provided, five of which are the ones discussed in Appendix
   B;struct client *t is the information structure of a client who sent the
   request; int. secret is the secret used for establishing a conference;
   int. n is the total clients; int. ul is the size of the frist unique
   subset; int. w is the weight. */
/* ------------------------------------------------------------------------ */
serve_request(struct client *t,int secret,int n,int ul,int w)
{
 char c;                    /* Used to indicate type of encryption */
 char buffer[MAXSTRING];    /* Used as buffer for communication */
 char temp[MAXSTRING];      /* Used as buffer for communication */
 char temp1[MAXSTRING];     /* Used as buffer for communication */
 char temp2[MAXSTRING];     /* Used as buffer for communication */
 char **data;               /* Used as buffer for client's names who want
                               to set a conference */
 int  i;
 int  number;               /* Indicates the number of clients who want to
                               set a conference */
 int  conference_held;      /* Checks if the request for asking a service
                               or communicating with clients  */
 struct client *cur,*block;

  /* Read the request sent by a client */
  readline(t->sockfd,buffer,MAXSTRING);

  if(((c = toupper(buffer[0])) == 'V'||c == 'B'||c == 'I') && buffer[1] == ' ')
    conference_held = TRUE;        /* The request is comm. command */
  else conference_held = FALSE;  /* Otherwise, the request asks a service */

  /* THE REQUEST IS ASKING A SERVICE FROM SERVER */
  if(conference_held == FALSE){
    decrypt('v',t->key,temp,buffer,FALSE);
    cur = root;

    /* IF THE REQUEST IS LIST ALL OF ACTIVE CLIENTS */
    if(strncmp(temp,".list",5) == 0){
      while(cur != NULL){
        if(cur->sockfd != UNUSED){
          if(cur->joint == FALSE)
            sprintf(temp,"Client: %10s, Level: %d, Join Conference: %s",
                    cur->id,cur->level,"No");
          else
            sprintf(temp,"Client: %10s, Level: %d, Join Conference: %s",
                    cur->id, cur->level,"Yes");
          encrypt('v',t->key,temp,buffer,FALSE);
          writen(t->sockfd,buffer,strlen(buffer));
        }
        cur = cur->next;
      }

    /* IF THE REQUEST IS JOIN  A CONFERENCE*/
    }else if(strncmp(temp,".join",5) == 0){
      /* Conference has not established yet */
      if(conference == FALSE){
        sprintf(temp,"Conference has not been established yet");
        encrypt('v',t->key,temp,buffer,FALSE);
        writen(t->sockfd,buffer,strlen(buffer));

      /* Otherwise */
      }else{
        /* Send the needed information to the client */
        sprintf(temp,".get %d %d",PRIME,secret);
        encrypt('v',t->key,temp,buffer,FALSE);
    writen(t->sockfd,buffer,strlen(buffer));

    /* Wait the client's response for confirmation */
    readline(t->sockfd,buffer,MAXSTRING);
    decrypt('v',t->key,temp,buffer,FALSE);
    i = atoi(temp);
    /* If the confirmation is valid */
    if(i == secret){
      t->joint = TRUE;

      /* Notify other clients who have joined a conference that the client
         joins the conference */
      cur = root;
      sprintf(temp1,"(%s,%d) joins the conference",t->id,t->level);
      while(cur != NULL){
        if(cur->sockfd != UNUSED && cur->joint == TRUE && cur != t){
          encrypt('v',cur->key,temp1,buffer,FALSE);
          writen(cur->sockfd,buffer,strlen(buffer));
        }
        cur = cur->next;
      }

      /* Send the client a positive response */
      sprintf(temp,"You can join the conference");
      encrypt('v',t->key,temp,buffer,FALSE);
      writen(t->sockfd,buffer,strlen(buffer));

    /* Otherwise, the confirmation is not valid */
    }else{
      sprintf(temp,"Sorry the secret is invalid ...");
      encrypt('v',t->key,temp,buffer,FALSE);
      writen(t->sockfd,buffer,strlen(buffer));
    }
  }

/* IF THE REQUEST IS LEAVE FROM A CONFERENCE */
}else if(strncmp(temp,".leave",6) == 0){
  /* The client has not joined any conference yet */
  if(t->joint == FALSE){
    sprintf(temp,"You are not joining any conference");
    encrypt('v',t->key,temp,buffer,FALSE);
    writen(t->sockfd,buffer,strlen(buffer));

  /* Otherwise */
  }else{
    /* Notify other clients who have joined a conference that the client
       leave the conference */
    i = 1;
    sprintf(temp,"(%s,%d) leaves the conference",t->id,t->level);
    while(cur != NULL){
      if(cur->sockfd != UNUSED && cur->joint == TRUE && cur != t){
        i += 1;
        encrypt('v',cur->key,temp,buffer,FALSE);
        writen(cur->sockfd,buffer,strlen(buffer));
      }
      cur = cur->next;
    }

    /* Send the client a positive response */
    sprintf(temp,"OK");
    encrypt('v',t->key,temp,buffer,FALSE);
    writen(t->sockfd,buffer,strlen(buffer));
    t->joint = FALSE;

    /* Check number of clients who leave the conference */
    i -= 1;
    /* If all of the clients leave the conference, conference is closed */
    if(i == 0 && Client == n){
      cur = root;

      /* Set all clients that they should receive a new info. about K */
      while(cur != NULL){
        if(cur->sockfd != UNUSED) cur->given = FALSE;
        cur = cur->next;
      }
      conference = FALSE;
      return YES;
    }
  }

/* IF THE REQUEST IS QUIT FROM SECRET CONFERENCE */
}else if(strncmp(temp,".quit",5) == 0){
  /* Notify all other clients that a client has quit */
  sprintf(temp,".quit %s %d",t->id,t->level);
  sprintf(temp2,"(%s,%d) quit from program",t->id,t->level);
  while(cur != NULL){
    if(cur->sockfd != UNUSED && cur != t){
      encrypt('v',cur->key,temp,buffer,FALSE);
      writen(cur->sockfd,buffer,strlen(buffer));
      encrypt('v',cur->key,temp2,buffer,FALSE);
      writen(cur->sockfd,buffer,strlen(buffer));
    }
    cur = cur->next;
  }

  /* Send the client a positive response & close the connection */
  sprintf(buffer,"OK");
  encrypt('v',t->key,buffer,temp,FALSE);
  writen(t->sockfd,temp,strlen(temp));
  close(t->sockfd);

  /* If all clients have quit, create a new K & other info. */
  if((Client -= 1) == 0){
    conference  = FALSE;
    Start       = TRUE;
  }
  /* Assign the client's structure as free structure */
  t->sockfd = UNUSED;
  return t->level;

/* IF THE REQUEST IS SET A CONFERENCE */
}else if(strncmp(temp,".conference",11) == 0){
  /* If # of clients made connections < n or conference does not exist */
  if(Client < n || conference == TRUE){
    if(Client < n)
      sprintf(temp,"# of Clients now are less than the total client(%d)",n);
    else sprintf(temp,"Conference has already exist");
    /* Send the client a negative response */
    encrypt('v',t->key,temp,buffer,FALSE);
    writen(t->sockfd,buffer,strlen(buffer));
    return FALSE;

  /* Otherwise */
  }else{
    /* Get the # of clients who want to set a conference & their names */
    strtok(temp," ");
    strcpy(temp,strtok('\0',"\n "));
    sscanf(temp,"%d",&number);

    /* Allocate buffer for storing the clients' names */
    if((data = calloc(number,sizeof(char *))) == NULL){
      printf("Fatal Error in calloc data\n");
      exit(1);
    }
```

```
for(i = 0;i < number - 1 && strcmp(temp,NULL) != 0;i++){
  if((data[i] = calloc(20,sizeof(char))) == NULL){
    printf("Fatal Error in calloc data[i]\n");
    exit(1);
  }
  strcpy(temp,strtok('\0',"\n "));
  if(strcmp(temp,NULL) != 0) sscanf(temp,"%s",data[i]);
}
if((data[i] = calloc(20,sizeof(char))) == NULL){
  printf("Fatal Error in calloc data[number]\n");
  exit(1);
}
/* Include the name of the client who send the request */
strcpy(data[i],t->id);

/* Check the subset of the clients who want to set a confer. */
i = confirm(number,data,n,ul,w);

/* If the subset is valid */
if(i == TRUE){
  sprintf(temp,".conference request:");
  for(i = 0;i < number;i++){
    strcat(temp,data[i]);
    strcat(temp," ");
    cur = root;

    /* Notify the corresponding clients that a client want them to set a
       conferece */
    while(cur != NULL){
      if(cur->sockfd != UNUSED && strcmp(cur->id,data[i]) == 0 &&
            cur != t){
        sprintf(temp2,"(%s,%d) asks you ",t->id,t->level);
        if(number >= 3) strcat(temp2," & others to join conference");
        else strcat(temp2,"to join conference");
        encrypt('v',cur->key,temp2,buffer,FALSE);
        writen(cur->sockfd,buffer,strlen(buffer));
        cur->joint = TRUE;
        break;
      }
      cur = cur->next;
    }
  }

  /* Send the notify and wait for their response */
  cur = root;

  while(cur != NULL){
    if(cur->sockfd != UNUSED && cur->joint == TRUE){
      do{
        encrypt('v',cur->key,temp,buffer,FALSE);
        writen(cur->sockfd,buffer,strlen(buffer));
        readline(cur->sockfd,buffer,MAXSTRING);
        decrypt('v',cur->key,temp1,buffer,FALSE);
      }while(strncmp(temp1,"OK",2) != 0);
    }
    cur = cur->next;
  }

  /* Server helps the subset to exchange their info. for obtaining the
     secret key K */
  cur = root;

  while(cur != NULL){
    /* Notify the subset of clients to exchange their info. */
    if(cur->sockfd != UNUSED && cur->joint == TRUE){
      sprintf(temp,".send need");
      encrypt('v',cur->key,temp,buffer,FALSE);
      /* Send to each client of the subset */
      writen(cur->sockfd,buffer,strlen(buffer));
      /* Wait for a response */
      readline(cur->sockfd,buffer,MAXSTRING);
      decrypt('v',cur->key,temp,buffer,FALSE);
      temp[strlen(temp)-1] = '\0';
      memset(temp1,'\0',strlen(temp1));
      strcpy(temp1,temp);

      /* Help each client to exchange their information about K */
      block = root;

      while(block != NULL){
        if(block->sockfd != UNUSED && block != cur &&
           block->joint == TRUE){
          encrypt('v',block->key,temp1,buffer,FALSE);
          /* Send to each client of the subset */
          writen(block->sockfd,buffer,strlen(buffer));
          /* Wait for a response */
          readline(block->sockfd,buffer,MAXSTRING);
          decrypt('v',block->key,temp,buffer,FALSE);
          temp[strlen(temp)-1] = '\0';
          memset(temp2,'\0',strlen(temp2));
          strcpy(temp2,temp);
          do{
            encrypt('v',cur->key,temp2,buffer,FALSE);
            /* Send to each client of the subset */
            writen(cur->sockfd,buffer,strlen(buffer));
            /* Wait for a response */
            readline(cur->sockfd,buffer,MAXSTRING);
            decrypt('v',cur->key,temp,buffer,FALSE);
          }while(strncmp(temp,"OK",2) != 0);
        }
       block = block->next;
      }
    }
    cur = cur->next;
  }

  /* Ask each client to send the value of K he/she obtained */
  cur = root;
  sprintf(temp1,".send secret");

  while(cur != NULL){
    if(cur->sockfd != UNUSED && cur->joint == TRUE){
      encrypt('v',cur->key,temp1,buffer,FALSE);
      /* Send the request */
      writen(cur->sockfd,buffer,strlen(buffer));
      /* Wait for a response from the corresponding client */
      readline(cur->sockfd,buffer,MAXSTRING);
      decrypt('v',cur->key,temp,buffer,FALSE);
      /* If K sent by the client is not valid */
      if((i = atoi(temp)) != secret){
        /* Send a negative response to each client in the subset */
        sprintf(temp,"Error in obtaining the secret key by %s",cur->id);
        block = root;

        while(block != NULL){
          if(block->sockfd != UNUSED && block->joint == TRUE){
            encrypt('v',block->key,temp,buffer,FALSE);
            writen(block->sockfd,buffer,strlen(buffer));
            block->joint = FALSE;
          }
          block = block->next;
        }

        break;
      }
    }
```

```
          cur = cur->next;
        }
      }

      /* Free buffer used for storing the subset of clients */
      for(i = 0;i <= number;i++) free(data[i]);
      free(data);

      if(i == FALSE || cur != NULL){
        /* If the confirmation of the subset is not valid */
        if(i == FALSE){
          /* Send a negative response */
          sprintf(temp,"# of clients to set a conference < the # required");
          encrypt('v',t->key,temp,buffer,FALSE);
          writen(t->sockfd,buffer,strlen(buffer));

        /* Otherwise, K sent by a client in the subset is not valid */
        }else{
          /* Notify all clients that Server send a new secret key info. */
          sprintf(temp,"Sending a new secret key ...");
          cur = root;

          while(cur != NULL){
            if(cur->sockfd != UNUSED){
              encrypt('v',cur->key,temp,buffer,FALSE);
              writen(cur->sockfd,buffer,strlen(buffer));
              cur->given = cur->joint = FALSE;
            }
            cur = cur->next;
          }
          conference = FALSE;
          return YES;
        }
      /* Setting a conference is successful */
      }else{
        /* Notify all clients that a conference is set */
        sprintf(temp1,"Conference may begin...");
        cur = root;

        while(cur != NULL){
          if(cur->sockfd != UNUSED){
            do{
              encrypt('v',cur->key,temp1,buffer,FALSE);
              writen(cur->sockfd,buffer,strlen(buffer));
              readline(cur->sockfd,buffer,MAXSTRING);
              decrypt('v',cur->key,temp2,buffer,FALSE);
            }while(strncmp(temp2,"OK",2) != 0);
          }
          cur = cur->next;
        }
        conference = TRUE;
      }
    }
  }
  /* THE REQUEST IS FOR COMMUNICATION COMMAND */
  }else{
    /* Direct the communication flow to all clients who joined a conference */
    cur = root;
    while(cur != NULL){
      if(cur->sockfd != UNUSED && cur->joint == TRUE && cur != t)
        writen(cur->sockfd,buffer,strlen(buffer));
      cur = cur->next;
    }
  }
  return FALSE;
}
```

```
/* -------------------------------------------------------------------- */
  /*MAIN PROCEDURE: procedure to run Server program; int. argc is the
  total arguments needed when Serve program is invoked; char **argv used
  as buffer for putting the arguments; an example of invoking the program
  is Server 5 2 1&. */
/* -------------------------------------------------------------------- */
main(int argc, char **argv)
{
 char buffer[MAXSTRING];

 int i;
 int n,w,u1;
 int secret_key;       /* Used for the secret key for establishing a
                          conference */
 int index1,index2;    /* Used for counting number of clients in the first
                          & second unique subsets */
 int sockfd;           /* Used for providing a connection to a client when
                          the client asks a connection request */
 int newsockfd;        /* Assigns new socket for further communication
                          with a client who successfully made a connection
                          to Server */
 int max_descr;        /* Indicates max. of descriptor that are used */
 int n_found;          /* Indicates a request from a client is found */
 int cli_len;          /* Used by network protocol */
 int mem_alloc;        /* indicates if memory allocations are needed or not */

 struct client        *cur;
 struct sockaddr_in   cli_addr, serv_addr;
 struct timeval       wait_time;
 fd_set               readset;

  /* If the # of arguments issued < the required */
  if(argc < 4){
   printf("USAGE: Server <total clients> <first level members> <weight>\n");
   printf("Example: Server 5 2 1\n");
   exit(1);
  }

 /* Get info. about the total # of clients, size of U1, and the weight */
 n  = atoi(argv[1]); u1 = atoi(argv[2]); w  = atoi(argv[3]);
 srand(getpid());
 index1 = 0; index2 = u1;

 /* Create all info. about the secret key K and get K */
 secret_key = creation_phase(u1,n,TRUE);

 /* OPEN A TCP SOCKET (AN INTERNET STREAM SOCKET) */
 if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  err_dump("server: can't open stream socket");

 /* INITIALIZE ALL INFO. OF SERVER FOR BINDING SERVER TO LOCAL ADDRESS */
 memset((char *) &serv_addr, 0, sizeof(serv_addr));
 serv_addr.sin_family      = AF_INET;
 serv_addr.sin_addr.s_addr  = htonl(INADDR_ANY);
 if(argc == 5) serv_addr.sin_port = htons(atoi(argv[1]));
 else serv_addr.sin_port = htons(DEFAULT_PORT);

 /* BIND SERVER LOCAL ADDRESS SO THAT A CLIENT CAN SEND DATA TO IT */
 if(bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
  err_dump("server: can't bind local address");

 /* Initialize all info. used for connections & for setting a conference */
 listen(sockfd, 5);
 max_descr          = sockfd;
 root               = NULL;
 wait_time.tv_sec   = 0;
 wait_time.tv_usec  = 0;
 Start              = TRUE;
 conference          = FALSE;
 Client              = 0;
 mem_alloc           = TRUE;

 for(;;){
   cli_len = sizeof(cli_addr);
   FD_ZERO(&readset);              /* Initialize readset */
   FD_SET(sockfd,&readset);        /* Put sockfd into readset */
   cur = root;

   while(cur != NULL){
       if(cur->sockfd != UNUSED) FD_SET(cur->sockfd,&readset);
       cur = cur->next;
   }
   /* WAITING FOR CONNECTION OR OTHER REQUEST FROM A CLIENT */
   if((n_found = select(max_descr + 1,&readset,NULL,NULL,&wait_time)) > 0){
     /* A CONNECTION REQUEST SENT A CLIENT */
     if(FD_ISSET(sockfd,&readset)){
       /* THE CONNECTION FAILS */
       if((newsockfd=accept(sockfd,(struct sockaddr *)&cli_addr,&cli_len)) < 0)
         err_dump("server:accept error");
       /* THE CONNECTION SUCCEEDS */
       else{
         /* CONNECTION REQUEST EXCEEDS THE REQUIRED NUMBER (n) */
         if(index1 == u1 && index2 == n){
           sprintf(buffer,"Server will not serve more than %d clients\n",n);
           writen(newsockfd,buffer,strlen(buffer));
           close(newsockfd);
         }
         /* If info. exchanged between Server & a client is success */
         if((i = info_exchange(newsockfd,n,u1,w)) != FALSE){
           Client += 1;
           if(newsockfd > max_descr) max_descr = newsockfd;
           if(i == 1) index1 += 1;
           else       index2 += 1;
           /* If the total # of clients is same as n, Server distr. all info. */
           if(index1 == u1 && index2 == n){
             if(Start == TRUE){          /* Indicates distr. is for 1st time */
               if(mem_alloc == TRUE){    /* Indicates memory alloc. is needed */
                 Start       = distribution_phase(TRUE,TRUE,n,u1,w);
                 mem_alloc   = FALSE;
               }else Start   = distribution_phase(FALSE,TRUE,n,u1,w);
             }else Start     = distribution_phase(FALSE,FALSE,n,u1,w);
           }
         }
       }
     }
     /* OTHER REQUEST SENT BY A CLIENT WHO ESTABLISHED THE CONNECTION */
     cur = root;
     while(cur != NULL){
       if(cur->sockfd != UNUSED){
         if(FD_ISSET(cur->sockfd,&readset)){
           if((i = serve_request(cur,secret_key,n,u1,w)) == YES){
             secret_key = creation_phase(u1,n,FALSE);
             Start      = distribution_phase(FALSE,TRUE,n,u1,w);
             break;
           }else if (i == 1) index1 -= 1;
           else if(i == 2)   index2 -= 1;
         }
       }
       cur = cur->next;
     }
   }
 }
}
```

```
/* ================================================================= */
/*Client.c contains seven procedures used for simulating a client
  in Secret Conference.                                               */
/* ================================================================= */
#include <pwd.h>      /* Header file for current working directory */
#include <netdb.h>    /* Header file for host server */
#include "define.h"   /* Contains all other header files */

#define LIMIT  1000   /* Used for obtaining private communication key */

typedef struct other{ /* Structure for saving other client's information */
  char      *name;    /* Name of other client */
  unsigned  level:2;  /* Indicates the group(unique subset) to which the other
                         client belongs */
   struct other *next;
}OTHER;

OTHER *root;

char Name[10];          /* The client's name */
char Secret[12];        /* The secret comm. key in a conference  */
char key[12];           /* Private comm. key used only by a client and Server */

int shadow;             /* The shadow for determining Secret from Y(x) */
int Level;              /* Indicates to the group (unique subset) to which the
                           client belongs */
int Prime;              /* Prime number used for determining Secret from Y(x) */
int Cypher;             /* Indicates cypher data is displayed on/off */
int Conference;         /* Indicates whether a conference exists or not */
int degree;             /* Degree of Y(x) polynomial */
int weight;             /* The weight of the members of the first group (unique
                           subset) over that of the second one */
int Im;                 /* Used for assigning index1 or index2 */
int indexx;             /* Index counter of other polynomial */
int index1;             /* Index counter of Z1(x) polynomial */
int index2;             /* Index counter of Z2(x) polynomial */
int mixed;              /* Indicates whether the clients who want to set a
                           conference are from both the first unique subset and
                           the second one or not */
int size;               /* Total Ims pooled by the clients who want toset a
                           conference */
int subset;             /* Subset of clients who want to set a conference */
int subset_done;        /* Indicates number of information still need to be
                           proccessed in obtaining Secret */

int **x;                /* Used as x parts of other polynomial */
int X[20];              /* Used as coeff. indexes of Y(x) polynomial */
int X1[10];             /* Used as x parts of Z1(x) polynomial */
int X2[10];             /* Used as x parts of Z2(x) polynomial */

double      **y;        /* Used for saving values of other polynomial */
long double Y[20];      /* Used to save values of coeffs. of Y(x) polynomial */
double      Z1[10];     /* Used for saving values of Z1 (x) polynomial */
double      Z2[10];     /* Used for saving values of Z2 (x) polynomial */

/* ------------------------------------------------------------------ */
  /* Procedure to obtain Prime and other using matrix operation discussed
  in Section 2.1; int. *A is used as x parts of a polynomial; double *B is
  used as coeff. parts of the polynomial; int. dim is dimension of matrix
  used.*/
/* ------------------------------------------------------------------ */
obtain_value(int *A,double *B,int dim)
{
 int i,j,k;              /* Temporary variables */
 int row;                /* Indicates a row in matrix */
 int col;                /* Indicates a column in matrix */
 int error;              /* Indicates an error occurs */
 double value;           /* Temporary variable */
double data;              /* Temporary variable */
double *temp1;            /* Temporary matrix */
double *temp2;            /* Temporary matrix */
double m[7][7];           /* Used for constructing matrix of x parts of the
                             polynomial */
long double inv[7][7];  /* Used as inverse matrix of m */

 /* Put x's values of a polynomial into matrix m */
 for(i = 0;i < dim;i++){
   for(j = 0;j < dim;j++){
     m[i][j]    = pow(A[i],j);
     inv[i][j] = 0;
     if(j == i) inv[i][j] = 1; /* Set inverse matrix as identity matrix */
   }
 }

 /* Apply the second method of finding inv. matrix described in Section 2.1 */
 error = FALSE;
 for(i = 0;i < dim && error == FALSE;i++){
   /* Step 1 of the method */
   if(m[i][i] == 0){
     printf("In Zero\n");
     col = i;
     row = i + 1;

     /* Find a row whose entry (i,i) is not zero */
     while(row < dim) if(m[row++][col] != 0) break;
     /* If there is such a row */
     if(row < dim){
       /* Swap all entries */
       temp1 = calloc(dim,sizeof(double));
       temp2 = calloc(dim,sizeof(double));
       memcpy(temp1,m[row],dim);
       memcpy(temp2,inv[row],dim);
       memcpy(m[row],m[i],dim);
       memcpy(inv[row],inv[i],dim);
       memcpy(m[i],temp1,dim);
       memcpy(inv[i],temp2,dim);
       free(temp1); free(temp2);
     /* Otherwise */
     }else{
       error = TRUE;
       printf("ERROR !!!\n");
     }
   }

   /* Proceeds to step 2 of the method */
   if(error == FALSE){
     row = col = i;

     /* Divide all entries in row i by entry (i,i) */
     while(row < dim){
       if(m[row][col] != 1){
         data = m[row][col];
         for(j = 0;j < dim;j++){
           m[row][j]    /= data;
           inv[row][j] /= data;
         }
       }
       row += 1;
     }

     /* Proceeds to step 3 of the method */
     row = col = i;
     for(j = 0;j < dim;j++){
       if(j != row && m[j][col] != 0){
         data = m[j][col];
```

```
        /* Subtract all entries in row j from the corresponding entries in row
           i */
        for(k = 0;k < dim;k++){
          m[j][k]   -= (data * m[row][k]);
          inv[j][k] -= (data * inv[row][k]);
        }
      }
    }
  }
 /* Repeat all steps starting from step 1 until i = dim */
 }
 value = 0;
 for(i = 0;i < dim;i++) value += (inv[0][i] * B[i]);
 return (error == FALSE) ? value : FALSE;
}

/* ---------------------------------------------------------------------- */
  /* Procedure to extract all information sent by Server which is required
  to  determine  Prime  and  Secret;  char.  *buf  contains  the  required
  information.*/
/* ---------------------------------------------------------------------- */
void distribute(char *buf)
{
 char c;                          /* Temporary variable */
 char temp[20];                   /* Used for obtaining info. from Server */
 char buffer[MAXSTRING];          /* Buffer used for communication */
 int i,j;                         /* Temporary variables */

  /* Iniliaze x parts of Z(x)s and Z(x)s */
  for(i = 0;i < 10;i++){
    X1[i] = X2[i] = 0;
    Z1[i] = Z2[i] = 0;
  }
  /* EXTRACT THE SHADOW & I_m FROM THE PACKET SENT BY SERVER */
  strcpy(buffer,buf);
  strtok(buffer," ");
  strcpy(buffer,strtok('\0',"\v"));
  sscanf(buffer,"%d %c",&shadow,&c); /* Get the shadow of K */
  if(c == ':'){                      /* Discard delimiter */
    strtok(buffer,":");
    strcpy(buffer,strtok('\0',"\v"));
  }else{
    strtok(buffer," ");
    strcpy(buffer,strtok('\0',"\v"));
  }
  if(c == ':' || Level == 2){       /* If the client is in U1 or U2 */
    strcpy(temp,strtok(buffer,"(,\n "));
    X1[0]  = atoi(temp);            /* Get x's value of I_m = (x,Z(x)) */
    strcpy(temp,strtok('\0',")"));
    Z1[0]  = atof(temp);            /* Get Z(x)'s value of I_m = (x,Z(x)) */
    if(c == ':') strcpy(buffer,strtok('\0',"\v"));
  }else{                            /* Client is in U1 & Case II of m occurs */
    X1[0]  = UNUSED;
    Z1[0]  = UNUSED;
  }
  if(c == ':' || Level == 1){       /* If the client is in U1, get other I_ms */
    for(j = 0;j < weight;j++){
      strcpy(temp,strtok(buffer,"(,\n "));
      X2[j]  = atoi(temp);          /* Get x's values of I_m = (x,Z(x))
      strcpy(temp,strtok('\0',")"));
      Z2[j]  = atof(temp);          /* Get Z(x)'s values of I_m = (x,Z(x)) */
    }
  }
  /* EXTRACT A SET Ci OF SOME COEFFICIENTS OF Y(X) FROM THE PACKET */
  for(i = 0;i < degree;i++) X[i] = Y[i] = 0;
  strcpy(buffer,strtok('\0',"\v"));
  strcpy(temp,strtok(buffer,"(,\n; "));
  while(strcmp(temp,NULL) != 0){
    if((i = atoi(temp)) < 0){
      strcpy(temp,strtok('\0',","));
      j = atoi(temp)-1;
    }else j = i-1;
    X[j] = I;                          /* Get the indices of the coeffs. */
    strcpy(temp,strtok('\0',")"));
    Y[j] = atof(temp);                 /* Get the values of the coeffs. */
    strcpy(temp,strtok('\0',"(,\n "));
  }
  Im = UNUSED;
}

/* --------------------------------------------------------------------- */
   /* Procedure to display the client information; WINDOW *win is used
   template to display the information; char *s contains the name of the
   client. */
/* --------------------------------------------------------------------- */
void info_display(WINDOW *win,char *s)
{
 int I;                                /* Used to locate the info. */

  i = strlen(s);
  /* Make the background of the info. black */
  wattron(win,A_REVERSE);
 /* Put the info. at the bottom part of the screen */
  mvwprintw(win,21,0,"Name: %s\t Level: %d\t ",Name,Level);
  if(Cypher == TRUE)
    mvwprintw(win,21,30+i,"Join Conference: %s\t Cypher : Yes\n",s);
  else mvwprintw(win,21,30+i,"Join Conference: %s\t Cypher : No \n",s);
  /* Make the background of other part back to normal */
  wattroff(win,A_REVERSE);
  wrefresh(win);
}

/* --------------------------------------------------------------------- */
   /* Procedure to process the Server reply; int. sockfd is a connection
   port used for communication with the Server; WINDOW *win is used as a
   template to display the needed information sent by the Server.*/
/* --------------------------------------------------------------------- */
void process_reply(int sockfd,WINDOW *win)
{
 char c;                  /* Indicates type of encryption scheme */
 char temp[20];           /* Used for storing some information */
 char buffer1[MAXSTRING]; /* Buffer for communication */
 char buffer2[MAXSTRING]; /* Buffer for communication */

 int i,j,k;               /* Temporary variables */
 int divided_coef;        /* Check whether any coeff. of Y(x) is divided into
                             several pieces or not */
 int level;               /* Used for checking other client's level */
 int for_conference;      /* Checks whether the reply is for a conference or
                             for any request sent before */

 double value1,value2;

 OTHER *prev,*cur;

  /* OBTAIN THE REPLY */
  readline(sockfd,buffer1,MAXSTRING);
  if(Cypher == TRUE) mvwprintw(win,20,0,"%s",buffer1);
  if(((c=toupper(buffer1[0]))=='V'||c == 'I'||c == 'B') && buffer1[1] == ' '){
    decrypt(c,Secret,buffer2,buffer1,TRUE);
    for_conference = TRUE;
  }else{
    decrypt('v',key,buffer2,buffer1,FALSE);
    for_conference = FALSE;
  }
```

```
/* THE REPLY IS AS A RESPONSE FROM SERVER */
if(for_conference == FALSE){

  /* THE REPLY IS INFORMATION DISTRIBUTION */
  if(strncmp(buffer2,".distribute",11) == 0){
    distribute(buffer2);
    sprintf(buffer1,"OK");
    encrypt('v',key,buffer1,buffer2,FALSE);
    writen(sockfd,buffer2,strlen(buffer2));
    mvwprintw(win,20,0,"Client receives information from Server ...\n\n");

  /* THE REPLY IS INFORMATION ABOUT ANOTHER CLIENT IS ACTIVE */
  }else if(strncmp(buffer2,".participate",12) == 0){
    sscanf(buffer2,"%s %s %d",buffer1,temp,&level); /* Get the info. */
    prev = NULL; cur = root;

    /* Allocate free structure for storing the new client's info. */
    while(cur != NULL){
      prev = cur; cur = cur->next;
    }
    cur = (OTHER *)malloc(sizeof(OTHER));
    if(cur == NULL){
      mvwprintw(win,20,0,"Fatal Error in malloc !!!\n\n");
      endwin(win);
      exit(1);
    }
    /* Allocate buffer for storing the new client's name */
    if((cur->name = malloc((strlen(temp)+1)*sizeof(char))) == NULL){
      mvwprintw(win,20,0,"Fatal Error in first malloc cur->name\n\n");
      endwin(win);
      exit(1);
    }
    /* Put the client's info. into the structure */
    strcpy(cur->name,temp);
    cur->level  = level;
    cur->next   = NULL;
    if(prev == NULL) root = cur;
    else prev->next = cur;

  /* THE REPLY IS A RESPONSE FOR JOIN REQUEST */
  }else if(strncmp(buffer2,".get",4) == 0){
    sscanf(buffer2,"%s %d %s",buffer1,&Prime,Secret);
    sprintf(buffer1,"%s",Secret);
    encrypt('v',key,buffer1,buffer2,FALSE);
    writen(sockfd,buffer2,strlen(buffer2));

  /* THE REPLY IS OTHER CLIENT'S REQUEST FOR SETTING A CONFERENCE */
  }else if(strncmp(buffer2,".conference request",19) == 0){
    /* Initialize some parameters */
    mixed = FALSE;
    subset = size = 0;
    /* Get the subset of clients who want to set a conference */
    strtok(buffer2,":");
    strcpy(temp,strtok('\0',"\n "));

    /* Get each client's name of the subset */
    while(strcmp(temp,NULL) != 0){
      if(strcmp(Name,temp) != 0){   /* If the name is not her/his-self name */
        cur = root;

        /* Check if the subset is from U1 & U2; get the size of the subset */
        while(cur != NULL){
          if(strcmp(cur->name,temp) == 0){
            if(cur->level != Level) mixed = TRUE; /* It is from U1 & U2 */
            if(mixed == TRUE){                    /* Get the size */
              if(cur->level == 1) size += weight;
              else size += 1;
        }else{                                      /* It is from U1 or U2 */
          if(X1[0] == UNUSED) size += weight;
          else size += 1;
        }
        subset += 1;
        break;
      }
      cur = cur->next;
    }
  }
  strcpy(temp,strtok('\0',"\n "));
}

/* Add the size of his/her-self to the other's */
if(mixed == TRUE){
  if(Level == 1) size += weight;
  else size += 1;
}else{
  if(X1[0] == UNUSED) size += weight;
  else size += 1;
}
subset_done = subset;
sprintf(buffer1,"OK");
encrypt('v',key,buffer1,buffer2,FALSE);
writen(sockfd,buffer2,strlen(buffer2));

/* SERVER ASKS THE CLIENT TO OBTAIN INFORMATION OF Prime & Secret FROM
   OTHER CLIENT */
}else if(strncmp(buffer2,".send need",10) == 0){
  /* Which Z(x) is used; Im = 1 for U1; otherwise for U2 & other subset */
  if(mixed == TRUE || (mixed == FALSE && X1[0] == UNUSED)) Im = 2;
  else Im = Level;
  divided_coef = 0;
  sprintf(buffer1,".need %d:",Im);
  /* Check if some coeffs. of Y(x) is in form of Φ = (x,θ(x)) */
  for(i = 0;i < degree;i++){
    if(X[i] <= 0){
      sprintf(temp,"%d ",i+1);
      strcat(buffer1,temp);
      if(X[i] < 0) divided_coef += 1;
    }
  }
  /* If so, allocate buffer for (x,θ(x)) */
  if(divided_coef > 0){
    x = calloc(divided_coef,sizeof(int *));
    y = calloc(divided_coef,sizeof(double *));
    for(i = 0;i < divided_coef;i++){
      x[i] = calloc(subset,sizeof(int));
      y[i] = calloc(subset,sizeof(double));
    }
    /* Put the coeffs. into the new buffer */
    k = 0;
    for(i = 0;i < degree;i++){
      if(X[i] < 0){
        x[k][0] = X[i];
        y[k][0] = Y[i];
        k       += 1;
      }
    }
    indexx = 1;
  }
  encrypt('v',key,buffer1,buffer2,FALSE);
  writen(sockfd,buffer2,strlen(buffer2));

/* THE CLIENT SENDS INFORMATION OF Prime & Secret TO OTHER CLIENT */
}else if(strncmp(buffer2,".need",5) == 0){
  sscanf(buffer2,"%s %d",buffer1,&i);
  sprintf(buffer1,".give ");

  /* Give the appropriate Im to the client who need it */
  if(i == 2 && Level == 1){
    for(j = 0;j < weight;j++){
      sprintf(temp,"(%d,%f) ",X2[j],Z2[j]);
      strcat(buffer1,temp);
    }
  }else{
    sprintf(temp,"(%d,%f) ",X1[0],Z1[0]);
    strcat(buffer1,temp);
  }

  /* Discard delimiter */
  strcat(buffer1,";");
  strtok(buffer2,":");
  strcpy(temp,strtok('\0',"\n "));

  /* Give the appropriate coeffs. of Y(x) to the client who need them */
  while(strcmp(temp,NULL) != 0){
    j = atoi(temp);
    if(X[j-1] > 0){
      sprintf(temp,"(%d,%f) ",X[j-1],Y[j-1]);
      strcat(buffer1,temp);
    }else{
      sprintf(temp,"(%d,%d,%f) ",X[j-1],j,Y[j-1]);
      strcat(buffer1,temp);
    }
    strcpy(temp,strtok('\0',"\n "));
  }
  encrypt('v',key,buffer1,buffer2,FALSE);
  writen(sockfd,buffer2,strlen(buffer2));

/* RESPONSE OF OTHER CLIENT IN GIVING INFORMATION OF Prime & Secret */
}else if(strncmp(buffer2,".give",5) == 0){
  if(subset == subset_done){  /* The client get this for the first time */
    if(Im == 1 || (Im == 2 && Level == 2)) index1 = 1;
    else index2 = weight;
  }
  /* Discard delimiter */
  strtok(buffer2," ");
  strcpy(temp,strtok('\0',"(,\n "));

  /* Put Im into the approriate place */
  while(strcmp(temp,";") != 0){
    i = atoi(temp);
    if(Im == 1 || (Im == 2 && Level == 2)){
      for(j = 0;j < index1;j++) if(X1[j] == i) break;
      strcpy(temp,strtok('\0',")"));
      if(j == index1){          /* Put Im into X1 & Z1(x) */
        X1[index1]  = i;
        Z1[index1]  = atof(temp);
        index1     += 1;
      }else size  -= 1;
    }else{
      for(j = 0;j < index2;j++) if(X2[j] == i) break;
      strcpy(temp,strtok('\0',")"));
      if(j == index2){          /* Put Im into X2 & Z2(x) */
        X2[index2]  = i;
        Z2[index2]  = atof(temp);
        index2     += 1;
      }else size  -= 1;
    }
    strcpy(temp,strtok('\0',"(,\n "));
  }

  k = i = 0;
  strcpy(temp,strtok('\0',"(,\n "));
  /* Put the coeffs. of Y(x) into the appropriate place */
  while(strcmp(temp,NULL) != 0){
    if((j = atoi(temp)) < 0){
      strcpy(temp,strtok('\0',","));
      X[atoi(temp)-1] = -1;
      x[k][indexx]    = j;
    }else X[j-1]      = j;
    strcpy(temp,strtok('\0',")"));
    if(j < 0){
      y[k][indexx]  = atof(temp);
      i = k; k += 1;
    }else Y[j-1] = atof(temp);
    strcpy(temp,strtok('\0',"(,\n "));
  }
  if(k > i)indexx += 1;
  subset_done     -= 1;

  /* If all the needed info. has beed received */
  if(subset_done == 0){
    /* Obtain the value of Prime m */
    if(Im == 1 || (Im == 2 && Level == 2))Prime= obtain_value(X1,Z1,size);
    else Prime = obtain_value(X2,Z2,size);
    if(Prime != FALSE && (Prime % 2) == 0) Prime += 1;
    k = 0;
    /* Obtain the coeffs. of Y(x) that are in form Φ = (x,θ(x)) */
    for(i = 0;i < degree;i++){
      if(X[i] < 0){
        Y[i] = obtain_value(x[k],y[k],subset);
        k   += 1;
      }
    }
    /* Obtain the secret key K */
    value1 = 0;
    for(j = 0;j < degree;j++){
      value2 = remainder(shadow,j + 1,Prime);
      value1+= Y[j] * value2;
    }
    sprintf(temp,"%f",value1);
    value1 = modf(value1,&value2);
    if(value1 >= .5) sprintf(Secret,"%d",atoi(temp) + 1);
    else sprintf(Secret,"%d",atoi(temp));
  }
  /* Notify the Server that the client has processed the info. */
  sprintf(buffer1,"OK");
  encrypt('v',key,buffer1,buffer2,FALSE);
  writen(sockfd,buffer2,strlen(buffer2));

/* AFTER OBTAINING Secret,THE CLIENT SENDS IT TO SERVER FOR CONFIRMATION */
}else if(strncmp(buffer2,".send secret",12) == 0){
  sprintf(buffer1,"%s",Secret);
  encrypt('v',key,buffer1,buffer2,FALSE);
  writen(sockfd,buffer2,strlen(buffer2));

/* REPLY IS INFORMATION THAT OTHER CLIENT QUIT FROM Secret Conference */
}else if(strncmp(buffer2,".quit",5) == 0){
  /* Get the info. of a client who has quit */
  sscanf(buffer2,"%s %s %d",buffer1,temp,&level);
  prev = NULL; cur = root;

  /* Search the corresponding structure of the client who quit */
  while(cur != NULL){
    if(strcmp(cur->name,temp) == 0) break;
    prev= cur;
    cur    = cur->next;
  }
  /* Discard the structure */
  if(cur != NULL){
      if(prev == NULL) root = cur->next;
      else prev->next = cur->next;
      free(cur);
    }

   /* SERVER SENDS INFORMATION THAT A CONFERENCE IS SET UP OR THE CLIENT IS
      GRANTED TO JOIN A CONFERENCE */
   }else{
     /* This is a response to 'join request' */
     if((strncmp(buffer2,"Conference may begin",20) == 0 && Im >= 1) ||
         strncmp(buffer2,"You can join the conference",27) == 0){
       Conference = TRUE;
       info_display(win,"Yes");
     }
     /* This is a response to the successful conference request */
     if(strncmp(buffer2,"Conference may begin",20) == 0){
       sprintf(temp,"OK");
       encrypt('v',key,temp,buffer1,FALSE);
       writen(sockfd,buffer1,strlen(buffer1));
     }
     wattron(win,A_BOLD);
     mvwprintw(win,20,0,"%s",buffer2);
     wattroff(win,A_BOLD);
   }

  /* THE REPLY IS FOR CONFERENCE */
  }else{
   wattron(win,A_BOLD);
   mvwprintw(win,20,0,"%s",buffer2);
   wattroff(win,A_BOLD);
  }
}

/* ------------------------------------------------------------------------ */
   /*procedure to exchange information between the client and Server by
   first setting a private comm. key used by both to exchange the
   information; int. sockfd is the client's connection port for
   communication with Server.*/
/* ------------------------------------------------------------------------ */
info_exchange(int sockfd)
{
 int i;                     /* Temporary variable */
 int a;                     /* Used as a base in a^private mod prime */
 int private;               /* Used as power in a^private mod prime */
 int prime;                 /* Used in a^private mod prime */
 int value1,value2;         /* Temporary variables */

 char data[20];             /* Used for storing info. */
 char buffer1[MAXSTRING]; /* Used for communication */
 char buffer2[MAXSTRING]; /* Used for communication */

 OTHER *prev,*cur;

  while((private = rand() % (LIMIT/10)) == 0); /* Get some value for power */

  readline(sockfd,buffer1,MAXSTRING);           /* Get the info. from Server */
  sscanf(buffer1,"%d %d %d",&a,&prime,&value1);

  i = remainder(value1,private,prime);          /* Get the private key */
  sprintf(key,"%d",i);

  while((value2 = remainder(a,private,prime)) == 0);

  /* Send the result back to the Server for confirmation */
  sprintf(buffer1,"%d\n\0",value2);
  writen(sockfd,buffer1,strlen(buffer1));
  sprintf(buffer1,"%d",value1);
  encrypt('v',key,buffer1,buffer2,FALSE);
```

```
writen(sockfd,buffer2,strlen(buffer2));
/* Get the corresponding response from the Server */
readline(sockfd,buffer2,MAXSTRING);
decrypt('v',key,buffer1,buffer2,FALSE);

/* If the confirmation is valid, info. exchange begins */
if(strncmp(buffer1,"OK:",3) == 0){
  /* Send the client's info. to the Server */
  a = buffer1[strlen(buffer1) - 2];
  sprintf(buffer1,"%s %d",Name,Level);
  encrypt(a,key,buffer1,buffer2,FALSE);
  writen(sockfd,buffer2,strlen(buffer2));

  /* Obtain the response from the Server */
  readline(sockfd,buffer2,MAXSTRING);
  decrypt('v',key,buffer1,buffer2,FALSE);
  /* Get the degree of Y(x) & the weight of U1 over U2 */
  sscanf(buffer1,"%d %d",&degree,&weight);

  strtok(buffer1,":");                        /* Discard delimiter */
  strcpy(data,strtok('\0',"(,\n "));

  /* Obtain the info. of other clients who have made connections to Server */
  while(strcmp(data,NULL) != 0){
    prev = NULL; cur = root;

    /* Allocate a free structure for each of the clients */
    while(cur != NULL){
      prev = cur;
      cur  = cur->next;
    }
    cur = (OTHER *) malloc(sizeof(OTHER));
    if(cur == NULL){
      printf("Fatal Error in malloc !!!\n");
      exit(1);
    }
    /* Allocate buffer for storing client's ID in each structure */
    if((cur->name = malloc((strlen(data)+1)*sizeof(char))) == NULL){
      printf("Fatal Error in malloc cur->name\n");
      exit(1);
    }

    /* Put the name & level of each corresponding client in the structure */
    strcpy(cur->name,data);
    strcpy(data,strtok('\0',")"));
    cur->level = atoi(data);
    cur->next  = NULL;
    if(prev == NULL) root = cur;
    else prev->next = cur;
    strcpy(data,strtok('\0',"(,\n "));
  }

  /* Notify the Server that all info. has been received */
  sprintf(buffer1,"OK");
  encrypt('v',key,buffer1,buffer2,FALSE);
  writen(sockfd,buffer2,strlen(buffer2));
  return TRUE;

 /* The confirmation is not valid */
 }else{
  fputs(buffer1,stdout);
  return FALSE;
 }
}
```

```
/* ---------------------------------------------------------------------- */
   /* Procedure to display help menu or Secret Conference information;
   WINDOW *w is used as a template to display the information.*/
/* ---------------------------------------------------------------------- */
void help(WINDOW *w)
{
  wmove(w,20,0);
  wrefresh(w);
  mvwprintw(w,20,0,"SECRET CONFERENCE:\n\n");
  mvwprintw(w,20,2,"Request Commands: \n");
  mvwprintw(w,20,3,"* Every command should begin with '.' as follows:\n\n");
  mvwprintw(w,20,4,".conference x C1 ... C(x-1) -->");
  mvwprintw(w,20,38,"Set a conference with other (x-1) clients\n");
  mvwprintw(w,20,4,".cypher on/off --> Display on/off the encrypted data\n");
  mvwprintw(w,20,4,".help  --> Display list of all commands\n");
  mvwprintw(w,20,4,".join  --> Join a conference if any\n");
  mvwprintw(w,20,4,".leave --> Leave a conference\n");
  mvwprintw(w,20,4,".list  --> List all clients that are active\n");
  mvwprintw(w,20,4,".quit  --> Exit from the Client Program\n\n");
  mvwprintw(w,20,2,"Communication Commands: \n\n");
  mvwprintw(w,20,4,"* To communicate with others who join a conference, ");
  mvwprintw(w,20,54,"data should be encrypted\n");
  mvwprintw(w,20,4,"* To encrypt the data, encryption type must be selected");
  mvwprintw(w,20,61,"by pressing\n");
  mvwprintw(w,20,6,"characater 'v', 'b', or 'i' before typing the data\n ");
  mvwprintw(w,20,6,"e.g.: 'b Hello ina, how are you?'\n\n");
  wmove(w,22,0);
  wclrtoeol(w);
  wrefresh(w);
}

/* ---------------------------------------------------------------------- */
   /*MAIN PROCEDURE: procedure to run Client program; an example of running
   the program is Client Nur 1.*/
/* ---------------------------------------------------------------------- */
main(int argc,char **argv)
{
 int i;                         /* Temporary variable */
 int c;                         /* Indicates the result of info. exchange */
 int connection;                /* Successful or fail connection to Server */
 int sockfd;                    /* Communication port address */
 int nfound;                    /* Indicates a response (from Server/stdin) */

 char Buf[MAXSTRING];           /* Used for communication */
 char buff[MAXSTRING];          /* Used for communication */
 char buffer[20];               /* Used for storing info. from Server */

 struct sockaddr_in  serv_addr; /* Used by network protocol */
 struct timeval      wait_time; /* Used by network protocol */
 fd_set              readset;   /* Used by network protocol */
 WINDOW              *win;      /* Used for displaying messages */

  if(argc < 3){
    printf("ERROR !!!\n");
    printf("Command should be as follows\n");
    printf("Client <name> <level>\n");
    printf("Example: Client Nur 1\n");
    exit(1);
  }

  /* INITIALIZATION */
  strcpy(Name,argv[1]);
  Cypher             = FALSE;
  Level              = atoi(argv[2]);
  wait_time.tv_sec   = 0;
  wait_time.tv_usec  = 0;
  Conference         = FALSE;
  srand(getpid());
```

```
/* INITIALIZE SERVER STRUCTURE FOR MAKING A CONNECTION TO SERVER */
memset((char *) &serv_addr,0,sizeof(serv_addr));
serv_addr.sin_family              = AF_INET;
serv_addr.sin_addr.s_addr         = inet_addr(SERV_HOST_ADDR);
if(argc == 4) serv_addr.sin_port = htons(atoi(argv[3]));
else serv_addr.sin_port           = htons(DEFAULT_PORT);

do{
  c = UNUSED;
  /* OPEN A TCP SOCKET (AN INTERNET STREAM SOCKET) */
  if((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0)
    printf("Client can't open stream socket\n");
  else{
      /* SENDS CONNECTION REQUEST TO SERVER */
      if((connection = connect(sockfd,(struct sockaddr *) &serv_addr,
           sizeof(serv_addr))) < 0) printf("Client can't connect to server\n");
      else c = info_exchange(sockfd);
  }

  /* If connection (connection) fail or info. exchange (c) fail */
  if(c == FALSE || sockfd < 0 || connection < 0){
    printf("Try again ? (y/n): ");
    gets(buffer);
    c = toupper(buffer[0]);
    close(sockfd);
    if(c == 'N') exit(1);
  }
}while(c == 'Y');

/* WINDOW INITIALIZATION */
win = initscr();                        /* Create a screen for input/output */
echo();                                 /* Display the input response */
info_display(win,"No");                 /* Display the client's info. */
wsetscrreg(win,0,20);                   /* Set area for output in the screen */
scrollok(win,TRUE);                     /* Enable scrolling */
idlok(win,TRUE);
help(win);                              /* Display help in the output area */
idlok(win,FALSE);                       /* Disable scrolling */
scrollok(win,FALSE);
wrefresh(win);

for(;;){
  FD_ZERO(&readset);                    /* Initialize readset */
  FD_SET(0,&readset);                   /* Put stdin descr. in readset */
  FD_SET(sockfd,&readset);              /* Put sockfd in readset */

  /* WAITING FOR RESPONSE EITHER FROM SERVER OR FROM STDIN */
  if((nfound = select(sockfd + 1,&readset,NULL,NULL,&wait_time)) > 0){

    /* RESPONSE FROM STDIN */
    if(FD_ISSET(0,&readset)){
      mvwgetstr(win,22,0,Buf);          /* Put the response into buffer */
      wmove(win,20,0);
      scrollok(win,TRUE);               /* Enable scrolling */
      idlok(win,TRUE);
      wrefresh(win);

      if(strcmp(Buf,"") == 0);          /* Do nothing */

      /* THE RESPONSE IS QUIT REQUEST */
      else if(strncmp(Buf,".quit",5) == 0){
        encrypt('v',key,Buf,buff,FALSE);
        writen(sockfd,buff,strlen(buff));
        readline(sockfd,buff,MAXSTRING);
        decrypt('v',key,Buf,buff,FALSE);
        if(strncmp(Buf,"OK",2) == 0){
          close(sockfd);
    endwin(win);
    exit(0);
  }

/* THE RESPONSE IS COMMUNICATION DATA */
}else if(Buf[0] != '.'){

  /* CONFERENCE DOES NOT EXIST YET */
  if(Conference == FALSE){
    mvwprintw(win,20,0,"Every command must be begun w/ '.'\n");
    mvwprintw(win,20,0,"E.g.  .list etc. or type '.help' for help\n\n");

  /* CONFERENCE EXISTS */
  }else{
    c = toupper(Buf[0]);

    /* ENCRYPTION TYPE IS NOT DETERMINED */
    if(Buf[1] != ' ' && !(c == 'V' || c == 'B' || c == 'I')){
      mvwprintw(win,20,0,"Determine encryption type, i.e., v or i ");
      mvwprintw(win,20,41,"or b, before communicating w/ others\n");
      mvwprintw(win,20,0,"E.g.  'b How is it goin' Alice?'\n\n");

    /* ENCRYPTION TYPE IS DETERMINED */
    }else{
      strtok(Buf," ");
      strcpy(buff,strtok('\0',"\v"));
      sprintf(Buf,"<%s,%d>: %s",Name,Level,buff);
      encrypt(c,Secret,Buf,buff,TRUE);
      writen(sockfd,buff,strlen(buff));
    }
   }

/* THE RESPONSE IS HELP COMMAND */
}else if(strncmp(Buf,".help",5) == 0) help(win);

/* THE RESPONSE IS CYPHER  COMMAND */
else if(strncmp(Buf,".cypher",7) == 0){
  memset(buffer,NULL,20);
  sscanf(Buf,"%s %s",buff,buffer);
  /* If the argument is 'on' or 'off' */
  if(buffer[0] != '\0'){
    if(strcmp(buffer,"on") == 0 || strcmp(buffer,"On") == 0 ||
        strcmp(buffer,"ON") == 0) Cypher = TRUE;
    else if(strcmp(buffer,"off") == 0 || strcmp(buffer,"Off") == 0 ||
             strcmp(buffer,"OFF") == 0) Cypher = FALSE;
    if(Conference == FALSE) info_display(win,"No");
    else info_display(win,"Yes");
  /* No argument, otherwise */
  }else mvwprintw(win,20,0,".cypher [on/off]\n");

/* THE RESPONSE IS NOT ANY REQUEST  COMMAND */
}else if(strncmp(Buf,".leave",6)!= 0&&strncmp(Buf,".list",5)!= 0&&
          strncmp(Buf,".join",5)!= 0&&strncmp(Buf,".conference",11)!= 0)
  mwprintw(win,20,0,"Invalid Command; Type '.help' for help\n\n");

/* THE RESPONSE IS OTHER REQUEST COMMAND */
else{
  i = UNUSED; memset(buff,NULL,strlen(buff));
  sscanf(Buf,"%s %d %s",buffer,&i,buff);

  /* THE RESPONSE IS CONFERENCE REQUEST BUT NOT VALID */
  if(strncmp(Buf,".conference",11) == 0 && (i == UNUSED ||
     strcmp(buff,NULL) == 0))
    mvwprintw(win,20,0,"Invalid Command; Type '.help' for help\n\n");

  /* OTHERWISE, SEND IT TO SERVER */
  else{
    encrypt('v',key,Buf,buff,FALSE);
```

```
          writen(sockfd,buff,strlen(buff));

          /* WAITING FOR SERVER'S REPLY FOR LEAVE REQUEST */
          if(strcmp(buffer,".leave") == 0 && Conference == TRUE){
            readline(sockfd,buff,MAXSTRING);
            decrypt('v',key,Buf,buff,FALSE);
            if(strncmp(Buf,"OK",2) == 0){
              Conference = FALSE;
              info_display(win,"No");
            }
          }
        }
      }
      scrollok(win,FALSE);              /* Disable scrolling */
      wmove(win,22,0);                  /* Locate cursor at the input area */
      wrefresh(win);                    /* Refresh the screen */

    /* THE RESPONSE IS FROM SERVER */
    }else if(FD_ISSET(sockfd,&readset)){
      scrollok(win,TRUE);               /* Enable scrolling */
      idlok(win,TRUE);
      process_reply(sockfd,win);        /* Call proced. to process the reply */
      scrollok(win,FALSE);              /* Disable scrolling */
      echo();
    }
    wmove(win,22,0);                    /* Locate cursor at the input area */
    wclrtoeol(win);                     /* Clear the input line */
    wrefresh(win);
  }
 }
}
```

VITA

Nur Hadisukmana

Candidate for the Degree of

Master of Science

Thesis: AN UNCHANGED SHADOW-BASED SECRET SHARING SCHEME

Major Field: Computer Science

Biographical:

Personal Data: Born in Jakarta, Indonesia, on July 23, 1963, son of Mr. D. Machdar Noor and Mrs. Hamida.

Education: Graduated from XXVII Public High School, Jakarta, Indonesia, in May 1982; received Sarjana Fisika degree with a major in Physics from the University of Indonesia, Jakarta, Indonesia in December 1988; completed the requirements for the Master of Science degree in Computer Science at Oklahoma State University, in December 1995.

Professional Experience: Systems Analyst and Programmer, Informatics Development Center, National Atomic Energy Agency, Jakarta, Indonesia, December 1988 to June 1990; Computer Networking Analyst, Informatics Development Center, National Atomic Energy Agency, Jakarta, Indonesia, June 1990 to June 1991.