Key Management Techniques for Dynamic Secure Multicasting: A Distributed Computing Approach

Madhu Sudana Rao Koneni

April 30, 2003

Blacksburg, VA

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

Dr. Donald Allison, Chairman

Dr. Lenwood Heath, Member

Dr. Ezra Brown, Member

Keywords: Dynamic Secure Multicasting, Chinese Remainder Theorem, Key Management Techniques, Public Key and Two Way Cryptography Systems

Key Management Techniques for Dynamic Secure Multicasting

Madhu S. Koneni

Abstract

Most of the Internet applications today require multicasting. For example, software updates, multimedia content distribution, interacting gaming and stock data distribution require multicast services. All of these applications require privacy and authenticity of the participants. Most of the multicasting groups are dynamic and some of them are large in number. Only those users who belong to the multicasting group should receive the information and be able to decrypt it. New users joining the group should receive information immediately but should not understand the information that was released prior to their joining. Similarly, if users leave the group, they should not receive any further information and should not be able to decrypt it. Keys need to be distributed to the users belonging to the current session and hence some kind of key management is required. Existing schemes for secure multicasting are limited to small and static groups. To allow large and dynamic groups to use the services of multicasting, some protocols have been developed: Multicast Trees, Spanning Tree, Centralized Tree-Based Key Management, Flat-key Management and Distributed Key Management. Some of these schemes are better than others with respect to the speed, memory consumption, and amount of communication needed to distribute the keys. All these schemes are limited in performance with respect to the speed, memory consumption, and amount of communication needed in distributing the keys.

In this thesis, a number of public and private key algorithms and key management techniques for secure and dynamic multicasting are studied and analyzed. The thesis is focused on the secure lock method developed by Chiou and Chen, using the Chinese Remainder Theorem. The protocol is implemented for a small group of users and its performance is studied. While, the secure lock method works well for a small group of users and the performance is degraded when the group grows in size. A protocol is proposed for a large and dynamic group, based on the idea of the Chinese Remainder Theorem. A performance study is carried out by comparing our proposed protocol with the existing multicasting protocols. The analysis shows that the proposed protocol works well for large and dynamic groups and gives significantly better performance.

Acknowledgements

I am greatly thankful to my advisor Dr. Donald Allison for all the help and advice on both the personal and professional fronts. I thank Dr. Lenwood Heath and Dr. Ezra Brown, for their valuable suggestions and corrections.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Sending a message to a well-defined group of users at the same time is known as multicasting. Most of the Internet applications today require multicasting. For example, software updates, multimedia content distribution, interactive gaming and stock data distribution require multicast services. All of these applications require privacy and authenticity of the participants. Hence cryptography plays a major role in the security aspects of multicasting. For example, consider, stock data distribution, which distributes stock information to a set of users around the world. It is obvious that only those who have subscribed to the service should get the stock data information. But the set of users is not static; the number of users keeps changing. New customers joining the group should receive information immediately but should not understand the information that was released prior to their joining. Similarly, if customers leave the group, they should not receive any further information. Hence, the problem is concerned with dynamic secure multicasting. To address this, group management is required. Some way is needed to create and destroy groups, and for processes to join and leave groups.

1.1 Research Statement

Secure multicasting requires membership control and transmission secrecy. Without membership control, everybody could join the group and there would be no secrecy. Once membership control is in place, the message should be transmitted in a secure way that only the members of the group access and understand the message. Having multiple unicast connections can solve the problem of multicasting. This approach requires the sender to encrypt n times and transmit the message n times, where n is the size of the group. Clearly, it is not going to scale well for large groups. To avoid the overhead of n encryptions and transmissions, and at the same time keep the secrecy, a single key known to all the members can be used to encrypt the message. This key is usually difficult to maintain without revealing to unauthorized people when the size of the group grows big. Also, another important issue is that the groups are not static; users join the group and leave the group at will. When the groups become large, membership changes very frequently and each change

requires a new key distribution. Hence, re-keying is a problem. Some kinds of key management techniques are needed to solve this problem. A number of protocols have been proposed so far and these work well for small and static groups. However, most of them fail to address the problems of scalability or the dynamic nature of large groups. Spanning tree protocol proposes the distribution of the key along a spanning tree generated between the members. It relies on trust in all members to forward the data without modification and does not handle group membership changes securely and efficiently. A second protocol, named Cliques, is proposed to improve the capability of a system to distribute the session keys in dynamic groups, but it has a scalability problem as the group manager has to perform n exponentiations for each membership change and messages get bigger. A third protocol, Iolus, proposes to divide a large group into small sub groups in order to reduce the number of members affected by a key change. It requires trust in the relay nodes and does not handle relay failures gracefully.

1.2 Research Goals

The overall goal of the thesis is to propose a viable protocol for dynamic secure multicasting with a large group of users. The proposed protocol satisfies the following requirements:

1. Work well in both small and large groups, as well as static and dynamic groups.
2. Achieve better efficiency compared to the existing protocols, in terms of the time taken to compute the encrypted and decrypted message, size of the encrypted message and the time taken to transmit the message.
3. Reduce overhead on the members of the group and distribute key computing responsibilities across the group.
4. Compute the new keys and transmit them only when required. Overall, reduce the number of times a new key is generated such that the network is not overloaded with keys.
5. Achieve the security level of public key algorithms.
6. Reduce the number of keys held by the members of the group.

Most of the research work is based on the paper ”Secure Broadcasting Using the Secure Lock”, by Chiou and Chen [3], where they suggest a secure lock protocol for broadcasting with the groups assumed to be static and small in size. The thesis also uses of centralized tree based key management, centralized flat key management, number theory and Unix multicast programming. The secure lock protocol is implemented in C and Unix using the Number Theory Library (NTL) [15].

The research focuses on studying various key management techniques for large and dynamic secure multicasting, and public and private key algorithms. Most of the applications using multicasting require privacy and authenticity. Hence, the first part of the thesis focuses on studying public and private key algorithms as cryptography plays a major role in secure multicasting.

1.3 Research Organization

Chapter 2 of the report is a literature review. It gives an introduction to multicasting, multicast addresses and Unix multicast programming. It also details the best-known public key cryptography algorithms, particularly the RSA, the Knapsack systems and the Rabin’s public key algorithms and their security and efficiency. Most of these public key algorithms are very slow compared to private key or symmetric key-encryption methods such as Data Encryption Standards (DES). The chapter also gives an overview of the two-way cryptography systems such as monoalphabetic system, substitution system, polybios checkerboard and affine system. The Diffie-Hellman key-exchange algorithm and some of the multicasting protocols like centralized tree-based key management, centralized flat-key management and distributed flat-key management are also described in this chapter.

The Secure Lock Protocol (SLP) proposed by Chiou and Chen is presented in Chapter 3. The chapter describes the secure lock approach, the mechanism to build the lock using the Chinese Remainder Theorem (CRT), the protocol using RSA and its implementation details. The proposed ways to handle joins and leaves of the group are also presented in this chapter. Then

finally, the problems with the approach and some of the solutions to the problems are described. The problem with Secure Lock Protocol is that it is efficient only when the number of users in the group is small, since the time to compute the lock and the length of the lock (hence the transmission time) are proportional to the number of users. The protocol works inefficiently when the number of users in the group is large, as it takes much longer time to compute the lock for more users. Hence, a Distributed Secure Lock protocol (DSLP) is presented in Chapter 4 based on the idea of secure lock and distributed computing so that the burden of computing the lock is distributed across the group. The chapter presents a detailed approach and the protocol, the algorithms, the Group Manager's role and Key Generation and Management. The relatively simple procedures of handling the leaves and joins of the group, the performance evaluation and the handling of security attacks are also explained in this Chapter.

Chapter 5 of the research presents the results achieved by implementing the secure lock protocol (SLP) and parts of the proposed DSLP protocol. Three main results are obtained, (i) the CRT computation results, (ii) decryption times taken by RSA and Rabin's Public key algorithms and (iii) the lengths of the keys for different sizes of the group. These results show that the proposed DSLP works more efficiently than SLP when the group size is large. It works equally well when the size of the group is small. The chapter ends with conclusions. The NTL Library is provided in the appendix A. A brief review of UNIX Multicast programming is provided in appendix B and the software is provided in appendix C.

Chapter 2: Literature Review

2.1 Multicasting

In some applications, widely separated processes work together in groups, for example, a group of processes implementing a distributed database system. It is frequently necessary for each member of these groups to send a message to all others in the group in a secure way. If one uses the point-to-point approach for secure multicasting of a message, then for each user in the group, the sender must perform encryption and send the cipher text separately. In the following diagram, Node 1 is the sender and the rest are the receivers in the network. Node 1 establishes four point-to-point connections and then sends the four different encrypted messages ($E_i(M)$) over those connections.

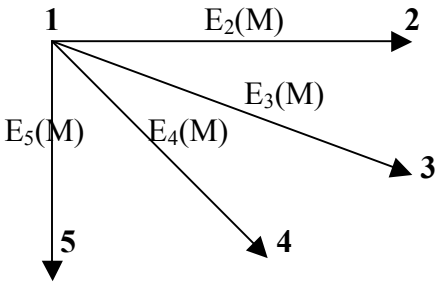


Figure 2.1: Transmission of the message M through 4 point-to-point connections

If every member in the group communicates with every other member, then the communication graph will look something like below.

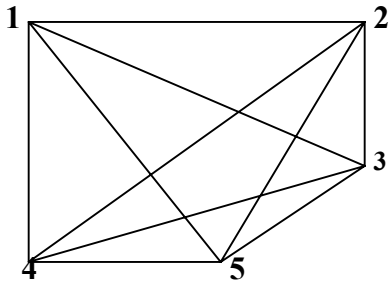


Figure 2.2: Communication graph when every member in the group communicates with every other member in the group.

In general, there will be n*(n-1)/2 individual point-to-point connections. This would work well as long as the group size is relatively small. If the group size is large, this strategy is inefficient. The other solution is to broadcast the message to all the members in the network. But the problem is that not all the members are interested in the message and the worst case could be where they are not supposed to receive the message. Hence, we need a way to send messages to well-defined groups that are large in size but relatively small compared to the network as a whole. Sending a message to such a group is called multicasting. Many Internet applications today use multicasting, for example: updating replicated and duplicated databases, transmitting stock quotes to multiple brokers, and handling digital conference telephone calls.

Implementing multicasting becomes a problem when the groups are large and dynamic (i.e., new users join the group and existing users leave the group). So, the main issues are group management and key management. Some of the other issues are congestion control and flow control. This thesis focuses on the security issues of group and key management.

2.1.1 Internet Multicasting

IP (Internet Protocol) supports multicasting, using class D addresses. Each class D address identifies a group of hosts. Twenty-eight bits are available for identifying groups; therefore over 250 million (2^{28}) groups can exist at a time in the Internet. Since IP is unreliable, when a packet is sent, not all the members in the group might get the packet. IP provides best effort delivery. Two kinds of multicasting addresses are supported by IP: Permanent addresses and temporary ones. Permanent groups are always there, so there is no need for group establishment. IP addressing is explained in the later part of this section.

Temporary groups need to be set up before they can be used for communication. A process can ask its host to join a particular group. Once the last process in the group leaves the group, the group doesn't exist anymore. Each host keeps track of the groups to which its processes currently belong. In Appendix C, a detailed discussion is provided on joins and leaves in UNIX. There are special multicast routers that implement multicasting. About every 60 seconds, each multicast router sends a hardware multicast to the hosts on its LAN asking them to report on the groups to

which their processes currently belong to. Each host sends back responses for all the class D addresses it is interested in. These query and response packets use a protocol called IGMP (Internet Group Management Protocol). Multicast routing is carried using spanning trees.

Multicast addresses

The range of IP addresses [16] is divided into "classes" based on the high order bits of a 32 bits IP address.

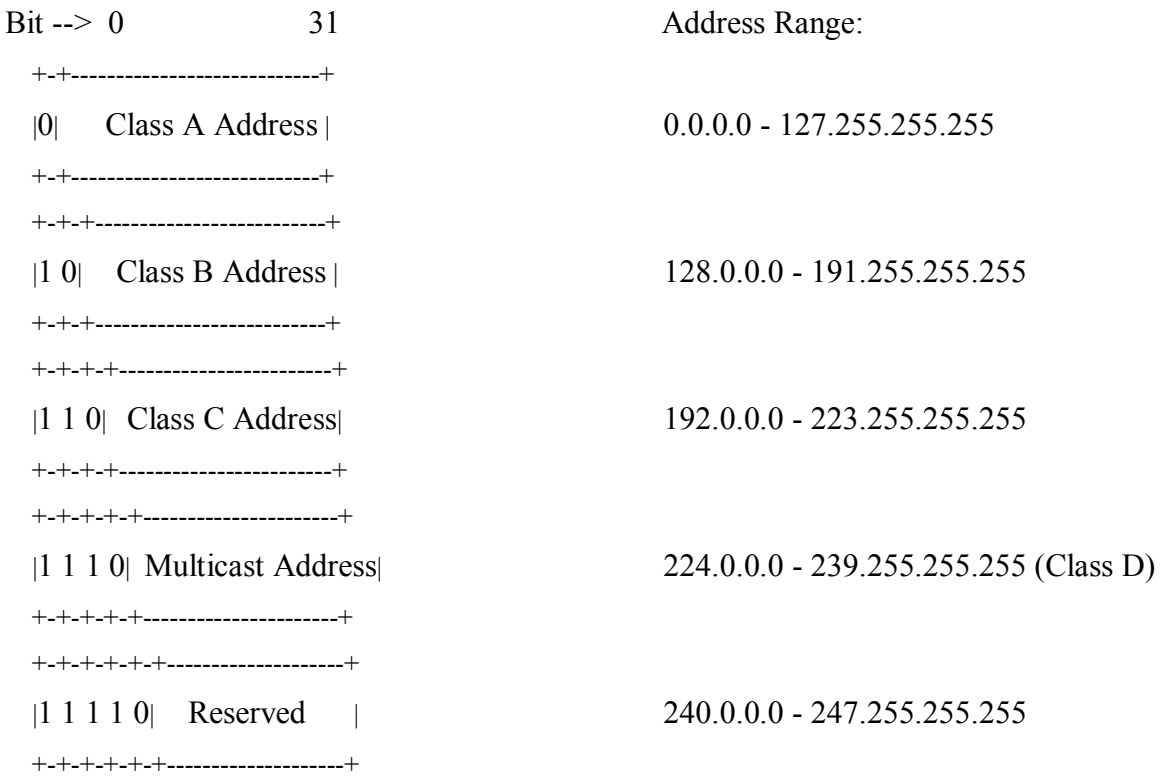


Figure 2.3: IP address classes

The TCP/IP standard for multicasting defines IP multicasting addressing and specifies how hosts send and receive multicast datagrams. IP multicasting uses the datagrams destination address to specify multicast delivery. IP multicasting uses class D addresses of the form shown below.

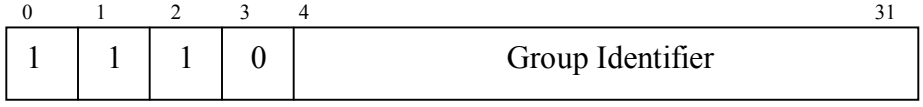


Figure 2.4: IP multicast address – class D address

The first four digits contain 1110 and identify the address as a multicast. Bits 4 thru 31 identify a particular multicast group. The group field does not contain a network address (since there is no single network to which all the hosts belong) like class A, B and C. When expressed in dotted decimal notation, multicast addresses range from range from 224.0.0.0 through 239.255.255.255. There is no single IP multicast address that refers to all the hosts in the Internet.

Some of the well-known multicast addresses are given below. Their addresses are fixed and cannot be changed.

224.0.0.1	All the hosts in the LAN
224.0.0.2	All routers in the LAN
224.0.0.4	All DVMRP (Distance Vector Multiple Routing) routers
224.0.0.4.1.1	All OSPF (Open Short Path First) routers
224.0.0.13.1	All PIM (Protocol Independent Multicast) routers

The range 224.0.0.0 through 224.0.0.255 is reserved for local purposes (as administrative and maintenance tasks) and datagrams destined for them are never forwarded to the multicast routers.

Multicasting Joins and Leaves

To join a multicast group, the process has to inform the operating system kernel which multicast groups it is interested in. When the multicast datagrams arrive, the kernel accepts only those datagrams that belong to the multicast group that the kernel's processes have joined. Any multicast-capable hosts join the *all-hosts* group (224.0.0.1) at start-up. So, for a process to receive multicast datagrams it has to ask the kernel to join the group and bind the port on which the datagrams were being sent. The UDP (User Datagram Protocol) layer uses both the destination address and port to demultiplex the packets and decide which socket to deliver them to. When a process is no longer interested in participating in a multicasting group, it informs the kernel that it wants to leave. The kernel will stop receiving the datagram packets belonging to that multicast group unless another process is interested in that group. This means that the host is

still participating in the multicast group. When all the processes on the host leave the group, then the kernel will stop receiving the datagram packets belonging to that multicast group.

2.2 Public Key Cryptography Algorithms

Whitfield Diffie and Martin Hellman [4] invented the concept of Public-key cryptography. The concept of Public-key is simple and elegant but has far-reaching consequences. There have been numerous Public-key algorithms proposed by different people. Only a few algorithms are both secure and practical. The reason why some of the algorithms are impractical is that either they have impractically large key or the cipher text produced is much larger than the plain text.

Public-key encryption is also known as asymmetric encryption since it is based on *one-way trap door functions*. In Public-key cryptosystems each entity has a public key 'e' and a private key 'd'. In secure systems, given a public key 'e', it is practically impossible to find out the private key 'd'. The public key need not be kept secret and can be known to anyone. But only the owner of the public key knows the corresponding private key. On the other hand, in symmetric-key algorithms, finding out the decryption key is trivial once the encryption key is known. Hence, a secure channel is needed for symmetric-key encryption schemes. The main objective of public-key encryption is to provide confidentiality. Public-key encryption methods are much slower than the symmetric-key encryption methods such as DES (Data Encryption Standards). Public-key algorithms are mostly used for encrypting small data items such as passwords, credit card numbers and PINs. They are not used for encrypting bulk data. In fact they are used to transport the key, which is ultimately used to encrypt the bulk data using symmetric-key encryption methods.

Three public-key algorithms have been studied in this thesis. They are the RSA, the Knapsack systems algorithm and the Rabin public-key algorithm. All of these algorithms are relatively slow.

2.2.1 RSA

The RSA public-key cryptosystem named after its inventors Rivest, Shamir and Adleman [5], is the most widely used public-key cryptosystem. It can be used for both confidentiality and digital signatures. Of all the public-key algorithms proposed, RSA is the easiest algorithm to understand and implement. From the time of its introduction, it has withstood significant cryptanalysis. Unfortunately, this has neither proved nor disproved RSA's security.

RSA's security is based on the intractability of the integer factorization problem. The public and private keys are functions of two large (more than 200 digits) random prime numbers. The problem of computing the RSA decryption key 'd' from the public key (n, e) is equivalent to the problem of factoring n. The algorithm for generating the keys is as follows:

1. Generate two large random (and distinct) prime numbers p and q**,** each roughly the same size.
2. Compute $n = pq$ and $\varphi = (p-1)(q-1)$.
3. Select a random integer e, $1<e<\varphi$, such that gcd $(e, \varphi) = 1$.
4. Using the extended Euclid's algorithm, compute the unique integer d, $1<d<\varphi$, such that $ed \equiv 1 \pmod{\varphi}$.
5. The public key is (n, e) and the private key is d.

RSA Encryption and Decryption Algorithms

The algorithm for encryption and decryption is explained below.

Encryption.

1. Obtain the public key (n, e).
2. Represent the plain text as an integer m in the interval [0, n-1].
3. Compute $c = m^e \bmod n$.
4. Send the cipher text c.

Decryption.

1. Use the private key d to get the plain text $m = c^d \bmod n$.

Security of RSA

The security of RSA depends on the difficulty of the problem of factoring large numbers. But it is neither proved nor disproved that factoring is the only means of breaking the security of RSA. However it is strongly believed that there is no other way of breaking RSA. A hacker knows the public key (n, e). To deduce the decryption key d, the hacker has to factor n. Factoring small numbers is not a big task but factoring large numbers is a challenging problem. The various algorithms on factoring are discussed in the following subsection. There has been no known efficient algorithm until now. As regards speed, RSA is very much slower than symmetric key algorithm. It is 1000 times slower than DES. That is why most practical systems use RSA only to exchange DES keys, and then use DES to encrypt everything else.

2.2.2 Knapsack Systems

Knapsack algorithms [5] obtain their security from the knapsack problem, an NP-complete problem. The knapsack problem can be stated as follows. A *knapsack vector* $A = (a_1,\ldots\ldots a_n)$ is an ordered n-tuple, $n\geq 3$, of distinct positive integers a_i. An instance of the knapsack problem is a pair (A, S), where A is a knapsack vector and S is a positive integer. A solution to an instance (A, S) is a subset of A whose sum is equal to S. A knapsack vector A is used to encrypt the plain text. The plain text is encoded as an integer, and the integer is represented as an n bits. All the integers in the knapsack, whose corresponding bit is 1, are added up. This is the cipher text. For example, let the knapsack be A = (1, 2, 3, 10, 20, 30, 40) and let us say that the plain text is represented in binary as 1010101. Then adding 1, 3, 20 and 40 will do encryption. Hence the cipher text is 64. Now, the problem of decryption includes finding the subset of A whose elements sum up to 64. Finding this is hard without knowing the knapsack itself. Even if the knapsack is known, the time required to solve this problem grows exponentially. The security of the Knapsack systems algorithm relies upon the NP-complete problem.

A solution can be found by exhaustive search of all the subsets of the knapsack. For a knapsack of *n* elements there are 2^n subsets and a search has to be carried out on all of them. This is good for small values of n but not for large values of n. When n is very large, the search through 2^n

subsets is unmanageable (the knapsack problem is an NP-complete problem). It shows that even the legal receiver has to find a solution to the NP-complete problem to obtain the plain text, which is not desirable. And also there should never be two solutions for the knapsack problem.

There are easy variations of knapsack problem. One of those of interest to us is a superincreasing set, which constitutes the knapsack problem. An n-tuple A = (a_1,……a_n) is called a superincreasing set if each number is greater than the sum of all its previous numbers. That is, $a_1 + a_2 + \ldots + a_{i-1} < a_i$ for all i. An integer *m* is chosen such that $m > \Sigma a_i$. Another integer t is chosen such that gcd (m, t) = 1. The choice of t should be such that another integer t^{-1} exists such that $tt^{-1} \equiv 1 \pmod m$. Now, the knapsack vector is multiplied by t and all the vector elements are reduced to modulo m. The resulting vector B = (b_1,…….b_n) is publicized as the encryption key. If the superincreasing knapsack vector is publicized as such, then the problem of decryption will become equally difficult for both the legal receiver and the cryptanalyst. Hence, The vector B is produced and publicized, as the encryption key and t, t^{-1} and m are kept secret. The actual algorithm is presented below.

Knapsack Algorithms

Algorithm: **Key generation for knapsack systems**.

1. An integer n is chosen.
2. Choose a superincreasing sequence (a_1,……a_n) and modulus M such that $M > a_1 + a_2 + \ldots + a_n$.
3. Select a random integer t, $1 \le t \le M-1$, such that gcd (M, t) = 1.
4. Compute $b_i = ta_i \pmod M$.
5. The public key is (b_1,…….b_n) and the private key is (M, t, t^{-1}).

Algorithm: **Encryption**

1. Obtain the public key (b_1,…….b_n).
2. Represent the plain text as a binary string of length n, X_1 X_2…….X_n.
3. Compute the integer $c = X_1b_1 + \ldots\ldots\ldots + X_nb_n$.
4. Send the cipher text.

Algorithm: **Decryption**

1. Obtain the private key (M, t, t^{-1}), the cipher text c and the public key (b_1,…….b_n).
2. Calculate $d = t^{-1}c \pmod{M}$.
3. Solve the superincreasing subsequence problem.

Algorithm: **Solving the superincreasing subsequence problem**

1. Obtain the knapsack vector A = (a_1,……a_n) and the cipher text *d*.
2. i = n

   while i ≥ 1{

   if d ≥ a_i then x_i = 1and d = d- x_i else x_i = 0.

   i = i-1.}
3. Return (x_1, x_2,……x_n).

All the above algorithms have been implemented and it was found that these were very easy to implement. Also encryption and decryption are much faster than RSA encryption and decryption.

Security of Knapsack

A pair of cryptographers [5] broke the knapsack cryptosystem. First a single bit of plain text was recovered. Then, Shamir showed that knapsacks could be broken in certain circumstances.

2.2.3 Rabin's Public key cryptosystem

Rabin's public-key encryption [4] scheme was the first provably secure system. The problem faced by the illegal receiver in order to recover the plain text is equivalent to the problem of factoring. Even though, it is widely believed that breaking the RSA system is equivalent to the problem of factoring, there exists no formal proof.

Rabin's PKC algorithms

Algorithm: **Key generation for Rabin's public-key cryptosystem**

1. Generate two large random numbers p, q, each must be of the form 4k+ 3 and roughly same size.
2. Compute n=pq.
3. The public key is n, and the private key is (p, q).

Algorithm: **Rabin's public-key encryption**

1. Represent the message as an integer m in the range {0, 1,……n-1}
2. Compute $c = m^2 \bmod n$.
3. Send the cipher text.

Algorithm: **Rabin's public-key decryption**

1. Find out the four square roots m1, m2, m3 and m4 of c modulo n.
2. One of the four square roots has to be the original text m.

Algorithm: **Finding the four square roots of c modulo n**

1. Using the extended Euclid's algorithm, find a and b satisfying ap + bq = 1
2. Compute $r = c^{(p+1)/4} \bmod p$.
3. Compute $s = c^{(q+1)/4} \bmod q$.
4. Compute x = (aps + bqr) mod n.
5. Compute y = (aps – bqr) mod n.
6. The four square roots are x, -x mod n, y and –y mod n.

Security of Rabin's cryptosystem

The problem faced by the passive adversary to recover the plain text is equivalent to the problem of factoring. Assuming that factoring is computationally intractable, Rabin's cryptosystem is provably secure.

Even though the system is provably secure against a passive adversary, the system succumbs to a chosen-cipher text attack.

The Rabin's cryptosystem is susceptible to hacker attacks similar to those on the RSA.

There is a drawback in the Rabin's cryptosystem. The receiver is faced with the problem of choosing the correct plain text from among the four possibilities. This problem is overcome by adding redundancy bits to the original text before encrypting. One of those recovered square roots m1, m2, m3, m3 will have redundant bits and the one with the redundant bits is the original plain text. If none of them has redundant bits, then the message can be rejected as a fraudulent message. There is yet another advantage of the redundancy bits. The problem of chosen-cipher text attack will not be there anymore. Rabin's encryption is an extremely fast operation as it only involves a singular modular squaring. Rabin's decryption is slower than encryption but comparable in speed to RSA's decryption.

2.3 Two-way Cryptography Systems

Before the invention of public key algorithms, two-way cryptography systems were most widely used. In a classical cryptosystem, the decryption key can easily be found once the encryption key is known, because the classical systems are based on symmetric functions. On the other hand, in public key cryptosystems, it is not trivial to determine the decryption key knowing the encryption key. Hence in classical cryptosystem, protection of the encryption key is very important. That is why the classical cryptosystems are called symmetric or two-way cryptosystems and the public key cryptosystems are called one-way or asymmetric cryptosystems. One such classical system is the Monoalphabetic system and is explained below.

2.3.1 Monoalphabetic system

There are various kinds of Monoalphabetic systems [5]. One of them is based on the concept that, in a good cryptosystem, the cryptosystem should be innocent looking. But, this is not a requirement at this time, since the message is in terms of 1s and 0s and it has no meaning at first sight to users. However, this idea was used in the past. The best way to make the message look

innocent is garbage-in-between. The idea is presented below. Richelieu used sheets of cardboard with holes. Only the letters visible through the holes are significant. Both the sender and the receiver should have the same sheets. For example the following is a cardboard with holes.

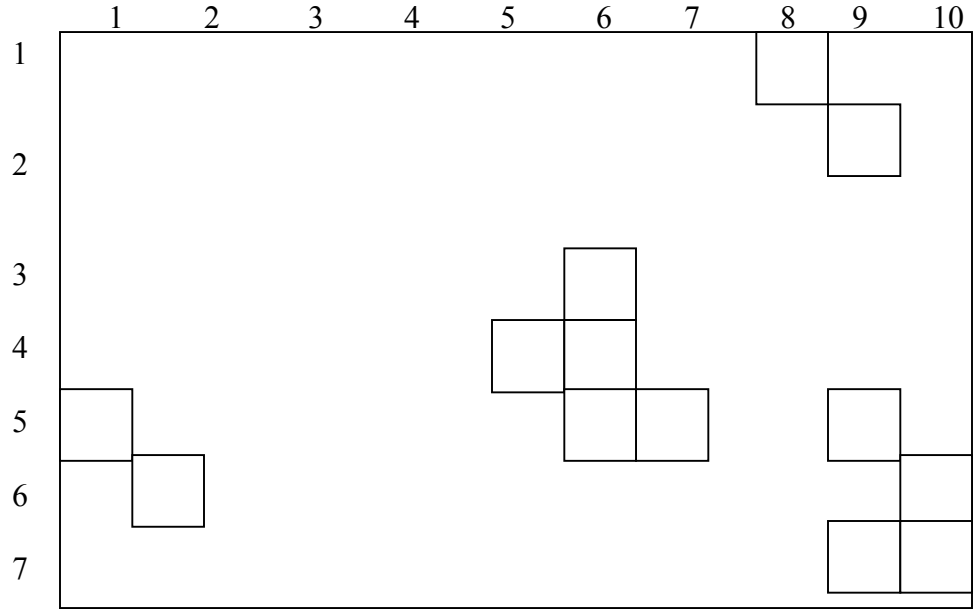


Figure 2.5: Cardboard with holes used in Richelieu cryptosystem

We can see from the above diagram that the holes are in the positions (1, 8), (2, 9), (3, 6), (4,5), (4, 6), (5, 1), (5, 6), (5, 7), (5, 9), (6, 2), (6, 10), (7, 9) and (7, 10).

Now, the following message looks like an innocent love letter.

I			L	O	V	E		Y	O	U
I			H	A	V	E		Y	O	U
D		E	E	P		U	N	D	E	R
M		Y		S	K	I	N		M	Y
L		O	V	E		L	A	S	T	S
F		O	R	E	V	E	R		I	N
H	Y		P	E	R	S	P	A	C	E

When the legal receiver gets this message, he/she uses the Richelieu cardboard and decrypts the message as YOU KILL AT ONCE. But, its security is poor. It does not take much effort from

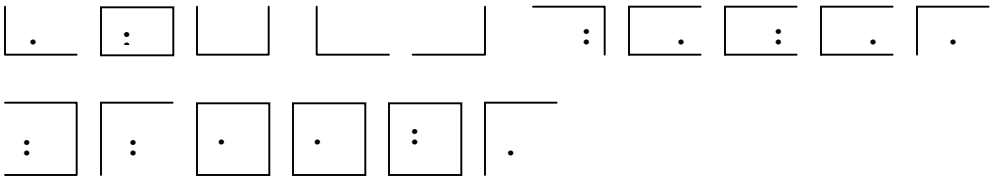
the cryptanalyst to decrypt the cipher encoded in this way. There are some old Monoalphabetic systems categorized as *substitution* and *permutation systems.*

2.3.2 Substitution system

In a *permutation system* [5], a permutation function is applied on the plain text. For example divide the plain text into blocks of three letters each and apply the permutation on each block separately. In each block the letters are moved one position ahead. For example, the plain text APPLYFORMULA will become PAPFLYMORAUL. But for an experienced cryptanalyst, it is not impossible to decipher the encrypted text. Hence, the system is not secure. In a *substitution system*, some entirely different character substitutes each letter in the English alphabet. For example, consider the following arrangement:

A:	**B:**	**C:**	**J.**	**K.**	**L.**	**S**	**T**	**U**
D:	**E:**	**F:**	**M.**	**N.**	**O.**	**V**	**W**	**X**
G:	**H:**	**I:**	**P.**	**Q.**	**R.**	**Y**	**Z**	

The lines surrounding each letter together with the dots indicate the substitutes for the letter. Thus, the plain text LET US GO FOR DINNER will be encrypted as



The crucial problem with this system is *key management*. The system breaks down once the correspondence between the plain text letters and the substitutes is known. Hence, the sender and the receiver have to memorize the key. The key should not be available in any form, anywhere.

2.3.3 Polybios Checkerboard

The oldest known cryptosystem is due to the Greek historian Polybios [5]. The Polybios checkerboard is shown below with the letter j omitted.

	A	B	C	D	E
A	A	B	C	D	E
B	F	G	H	I	K
C	L	M	N	O	P
D	Q	R	S	T	U
E	V	W	X	Y	Z

Figure 2.6: Polybios checkerboard

The two letters corresponding to the column and the row in which that letter lies will represent each letter. Thus, BB will represent G and H will be represented by BC. For example, the plain text LET US GO FOR DINNER will be encrypted as CAAEDDDEDCBBCDBACDDBADBDCCCCAEDB.

2.3.4 Affine System

An *affine cryptosystem* [5] is determined by two integers 'a' and 'b', where $0 \leq a, b \leq 25$ and, 'a' and 26 are relatively prime. Each letter $\propto$ will be substituted by $a\propto+b$ modulo 26. The requirement of 'a' and 26 being relatively prime ensures that the mapping $a\propto+b$ is one-to-one. Usually the letters are encoded into integers before applying the encryption function, with A having the value 0 and Z having the value 25. For example, if a = 3 and b = 5 then the letter Y with numerical encoding of 24 will have 25 (Z) as its encrypted value. For decrypting, the inverse of 'a' modulo 26 has to be found by applying Extended Euclid's algorithm. Then $a^{-1}\propto$ - b would give us the actual value. There are 12 possible values of 'a' since the number of relative primes of 26 which are less than 26 are 12. And there are 26 possible values for b. Totally 12*26

– 1(since the case of a = 1, b = 0 is excluded) = 311 possible keys for an affine system. Checking through all the 311 keys is easy and hence cryptanalysis is easy.

2.4 Diffie-Hellman Key-Exchange Algorithms

Diffie-Hellman [10] was the first public-key algorithm ever invented. Its security is based on the difficulty of calculating discrete logarithms in a finite field. Diffie-Hellman can be used to distribute the keys but not to encrypt and decrypt. Alice and Bob can use this algorithm to generate a secret key.

Alice and Bob agree on a large prime n and another integer g, such that g is primitive mod n. These two integers need not be kept secret. The following is the protocol followed by Alice and Bob.

1. Alice chooses a random large integer x and sends Bob
   $X = g^x \bmod n$
2. Bob chooses a random large integer y and sends Alice
   $Y = g^y \bmod n$
3. Alice computes
   $k = Y^x \bmod n$
4. Bob computes
   $l = X^y \bmod n$

Both k and l are equal to $g^{xy} \bmod n$. No one else other than Bob and Alice can compute the number $g^{xy} \bmod n$. Unless the discrete logarithm is computed and x (or y) is recovered, the secret key k (or l) cannot be computed. Now, Alice and Bob can use this private key to communicate the messages. The security of this approach depends on the size of the numbers g and n, since the security depends on the difficulty of factoring numbers the same size as n. This algorithm can be easily extended to work with three or more people. However, the problem with this approach is that every user in the system has to do the costly computation of finding the secret key k.

2.5 Multicasting Protocols

2.5.1 Centralized Tree-Based Key Management

In this centralized approach, there is only one manager (centralized), where the keys are centrally maintained and the users are registered. The keys are held in a binary tree. To store the keys, any tree of arbitrary degree is used. The participants are represented by the leaves in the tree. The following example depicts a fully balanced, complete binary tree with a maximum of 16 members and a depth of 4.

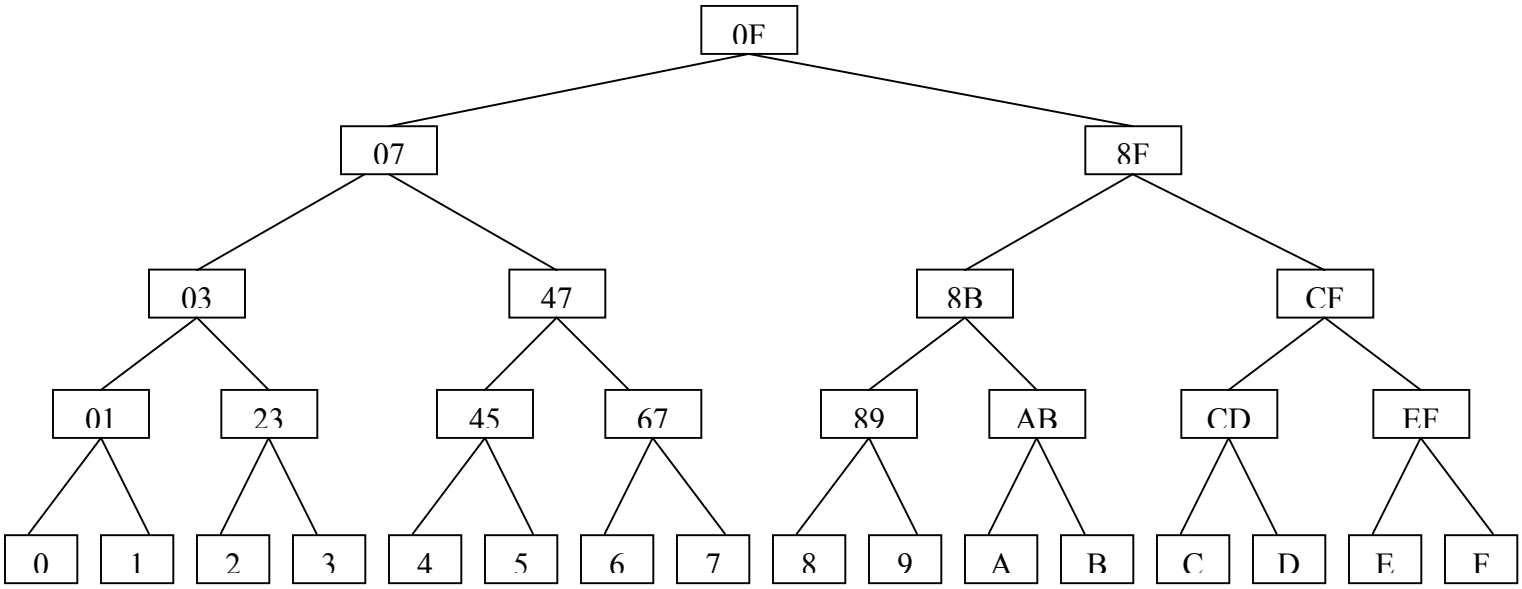


Figure 2.7: Binary hierarchy of keys. Labels in hexadecimal define the range of participants knowing this key [11].

The important observations to be made in this method are:

1. Each user has a shared secret established with the group manager (GM), who stores that information in the leaf associated with the user and uses it whenever a truly private communication is required with the user. This is called lowest level key encryption key (KEK).

2. The intermediate nodes in the tree do not correspond to any entities, but only hold keys for a hierarchy of virtual sub-groups of different sizes and are used to effectively communicate the new keying material when the membership of the group changes.
3. Each user has a subset of keys from the tree, the keys that are in the path from the leaf to the root, which is used as Traffic Encryption Key (TEK). These intermediate key encryption keys (KEKs) are used if a message should be understood by a subset of the group. For example, a message encrypted with KEK 03 is understood by the participants 0,1,2 and 3. Others will not be able to decrypt the message.

Joins and leaves are handled in the following way.

1. On a join operation, the participant's key manager unicasts its request to the group manager, who checks with admission control and assigns an ID, where the participant's key is stored. If, the ID is 4, then the key will be stored at the leaf 4 in the binary tree.
2. The ID will give the traffic encryption key (TEK) in the tree. For example, if the ID is 4, then the intermediate key encryption keys are 45, 47, and 07.

When an existing user leaves the group,

1. The group manager sends out a message with new keying material, which can only be decrypted by all the key managers of the remaining participants and frees the slot used by the leaving participant, making it available for reuse at the next join.
2. Let us say that '4' is leaving. The keys that it knew (key encryption keys 45, 47, 07 and the Traffic Encryption key 0F) are to be changed in such a way that 4 cannot acquire the new keys. This can be done by encrypting all the new intermediate node (from the leaf node of the leaving participant to the root node) keys with appropriate underlying node or leaf keys. When 4 leaves, the new key encryption key 45_{new} needs to be received by 5, 47_{new} needs to be received by 5, 6 and 7, 07_{new} needs to be received by 0….3, 5…..7 and the new traffic encryption key $0F_{new}$ needs to be received by every participant except 4. This is achieved by doing the following.
    a. 45_{new} is encrypted by 5, the only recipient in need of it and sent only once.

b. 47_{new} is encrypted twice, once by the existing 67 key and once by the new key 45_{new} and the two copies are sent.
c. 07_{new} is encrypted twice, once by the existing 03 key and once by the new key 47_{new}.
d. OF_{new} is sent twice encrypted once by the existing key 8F and the new key encryption key 07_{new}.

So, if the depth of the tree is W and there is a single leave, then 2W-1 keys are being sent out. A single message with all the new keys can be multicasted to the participants and the processing of this multicast message by the receivers will take at most W decryption operations, with an average of less than 2 decryptions.

2.5.2 Centralized Flat Key Management

Instead of assigning the keys and distributing them in a hierarchical tree-based fashion, they can also be assigned in a flat fashion. This is called Centralized Flat Key Management [11]. This reduces the database requirements, and the sender doesn't have to store the information about all the participants. The data structure held by the Group Manager is a simple table with 2W+1 entries. One entry holds the current TEK and the other 2W entries hold KEKs. Here W is the number of bits in the participant ID, which is normally equal to its transport layer or its network address. For each bit in the network address, two keys are available. Each participant knows W of these keys depending on the value of single bits in its address.

The results are similar to those of Tree-based control, but the key space is much smaller. For an ID of length W bits, only 2W+1 keys are needed irrespective of the number of participants in the group. The number of participants is limited to 2^W. This approach also keeps the size of the change messages small. Besides reducing the storage and communication needed, this approach has the advantage that nobody needs to keep track of who is currently a member, yet the Group Manager is still able to expel an unwanted participant.

To join, a participant contacts the Group Manager who assigns a unique ID and keys corresponding to the IDs bit/value pairs to the participant. All keys known to the leaving participant (TEK and W KEKs) are to be replaced in a way intractable to the leaving user, but easily computable for all remaining participants. The Group Manager sends a multicasting message with two parts: First, it contains new TEK encrypted with each of the valid KEKs and second, it contains a new replacement KEK encrypted with both the old KEK and the new TEK for each of the invalid KEKs.

2.5.3 Distributed Flat Key Management

There is a danger of implosion and the existence of a single point of failure. The solution is to completely distribute [11] the key database of the centralized flat approach such that all participants are created equal and no one has complete knowledge. There is no dedicated Group Manager; collaboration of multiple participants is required to propagate changes to the whole group. Each participant only holds the keys corresponding to its ID, as in centralized flat approach. The IDs are generated uniquely in a distributed way, as there is no dedicated Group Manager.

This scheme is the most resilient to network or node failures due to its inherent self-healing capability, but is more vulnerable to inside attacks than others. The approach can be considered stronger against active attacks.

Chapter 3: Secure Lock Protocol (SLP)

3.1 Secure Lock

3.1.1 Introduction

The main property of a multicast channel is that a single transmission from a sender should be simultaneously received by multiple users. Most of the time the transmitted messages are secret and should be received and understood by the intended receivers only. Thus, security is the main issue. There are many approaches proposed which work well when the size of the group is small and static. These approaches do not scale well when the groups are large and change dynamically. In the following section, secure multicasting using the secure lock approach proposed by Chiou and Chen [3] is presented. In the later sections, the design of the multicasting system using alternate protocols and its implementation are presented followed by the examination of the system for a large and dynamically changing group.

Secure broadcasting using the secure lock based on the Chinese Remainder Theorem has been proposed by Chiou and Chen [3]. The objectives of this thesis are

1. Make use of their idea to implement a secure multicasting system for a small set of users in a Local Area Network.
2. Design the system to manage the leaves and joins of the members.
3. Propose a viable protocol for a multicast system where the groups are large and dynamically varying.

The system for a small group of users in a LAN is implemented here using different encryption techniques and the results are presented in Chapter 5.

3.1.2 Secure Lock Approach

Assume that there is a group of n users in a network. Let them be denoted as $u_1, u_2, \ldots\ldots, u_n$. In order to send the message M to the group, it has first to be encrypted before it is transmitted. Since there are n different users, it will be advantageous if there exists a method that can securely send the deciphered session key together with the cipher text of M. Since the message is encrypted by the encrypt session key, only one copy of the cipher text needs to be sent, and since the session key is selected for each transmission, no additional keys need to be kept secret. Based on this idea, Chiou and Chen proposed an approach in which the lock is superimposed onto the front of the sent-out message to lock the deciphering session key, and only matches with keys of the users in the group. The format of the sent-out message looks like the following.

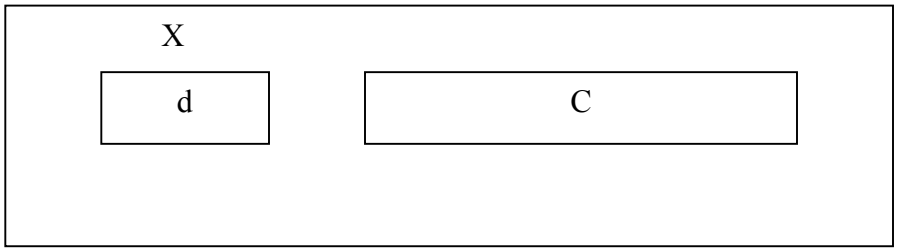


Figure 3.1: Format of the sent-out message in secure lock protocol

where

X is the lock

d is the deciphering session key; i.e., $M = D_d(C)$

C is the cipher text of the secret message M; i.e., $C = E_e(M)$

e is the encryption session key

The lock X should satisfy the following properties:

1. Users of the group can only open the lock since the lock contains the information about the decryption session key d.
2. The lock should depend only on d and should not disclose any information about the sender.

Thus the lock must functionally depend on e and d. The lock is constructed using the Chinese Remainder Theorem as discussed below.

3.1.3 Construction of the Lock

The lock is constructed based on the Chinese Remainder Theorem [10]. The theorem is as follows:

Let p and q be positive integers that are relatively prime and let a and b be any two integers. Then there is an integer N such that

$$N \equiv a \pmod{p}$$

and

$$N \equiv b \pmod{q}.$$

Moreover N is uniquely determined modulo pq. The generalized Chinese Remainder Theorem is as follows.

Given a set of simultaneous congruences

$$x \equiv a_i \bmod (m_i)$$

for $i = 1,\ldots, r$ and for which the m_i are relatively prime, the solution of the set of congruences is

$$x = a_1 b_1 M/m_1 + \ldots\ldots + a_r b_r M/m_r \pmod{M},$$

where

$$M \equiv m_1 m_2 \ldots . m_r$$

and the b_i are determined from

$$b_i M/m_i \equiv 1 \pmod{m_i}$$

The construction of the lock is as follows.

Let d be the deciphering session key

R_i be the cipher text of d enciphered with u_i's encryption key e_i

i.e., $R_i = Ee_i(d)$

and let N_1, N_2,……. N_n be pair wise relatively prime numbers, which are publicly known in the system. N_i is associated with user u_i and is greater than R_i. Then, we have the following set of congruences:

$X \equiv R_1 \bmod N_1$

$X \equiv R_2 \bmod N_2$

.

.

$X \equiv R_i \bmod N_i$ for all users u_i

. for all R_i, $R_i \leq N_i$

.

$X \equiv R_n \bmod N_n$ Where $R_i = Ee_i(d)$

A common solution X is found from the above equations using the Chinese Remainder Theorem. Once the common solution X is found, each user u_i can compute R_i from X (R_i = X % N_i). Therefore, u_i can decipher R to get d, the deciphering session key.

3.1.4 Performance of the Lock

The lock obtained above is secure because

1. Each of the congruences is associated with a user in the group, and to obtain d the user needs the associated deciphering session key d_i to decipher R_i to open the lock X. Therefore, only the users in the group have the key to open the lock.
2. Since $R_i = Ee_i(d)$, R_i is functionally dependent on d, the session deciphering key.
3. Each user only needs to know his own secret key. Hence there is no need for keeping extra keys secret.

In summary, sending the lock along with the cipher text has the following advantages:

1. Only one copy of the cipher text is sent, reducing the overhead on the sender and the network
2. For each user, there is no need to keep extra keys secret (other than his own secret keys).

3.2 Secure Lock Protocol

Based on the above idea, protocols are designed for a multicast system using the public key algorithm RSA and Rabin's public key algorithm. First, the protocol based on RSA is described. The protocol based on Rabin's public key algorithm is similar to the protocol based on RSA.

RSA based protocol

- Assume that there are n users in the system. Let them be denoted by $u_1, u_2, \ldots\ldots, u_n$. Each user u_i is associated with an integer N_i.
- Let $N_1, N_2,, \ldots Nn$ be pair wise relatively prime numbers, and are known publicly in the system.
- Each user u_i has a public key (e_i, n_i) and a private key d_i.
- Let (e, n) be the session encryption key and d be the session decryption key chosen by the sender.

The following structure is used to send the message:

```
struct message {
        char *X;        //Lock
        char *CKD;      //Verification code
        char *msg;      //Cipher Text
}
```

The sent out message looks like the following.

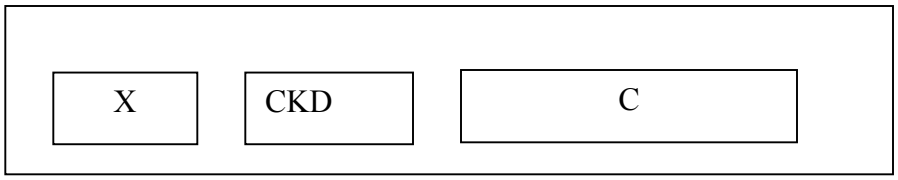


Figure 3.2: Format of the sent-out message in secure lock protocol

where

X is the solution of the congruences obtained using the Chinese Remainder Theorem, $X \equiv E(n_i, e_i, m) \bmod N_i$, for all users u_i, where E is the RSA encryption function discussed earlier and m = d.

CKD is the cipher text of d encrypted using e

i.e., CKD = E (n, e, d)

C is the cipher text of the plain text secret message M

i.e., M = E (n, e, M)

3.2.1 Sender's Encryption Algorithm

Input: The secret message M, the public keys (n_i, e_i) and the prime numbers N_i

Output: Secure Lock X, Encrypted text C and CKD.

1. Choose the session encryption key (n, e) by generating the large random prime numbers p and q using NTL. The session decryption key d is calculated.
2. Calculate $R_i = E(n_i, e_i, d)$ for all users u_i in the group.
3. Calculate the lock $X \equiv R_i \bmod N_i$ using the Chinese Remainder Theorem.
4. Encrypt the secret message M by calling the encryption function E (n, e, M) and store it in C.
5. Convert the ZZ integers X, CKD and C into bytes using BytesFrom (ZZ) function (see Appendix A).
6. Fill the structure message with the converted bytes.
7. Multicast the structure message.

3.2.2 Receiver's Decryption Algorithm

1. Receive the message Msg from the sender.
2. Obtain the ZZ integers X, CKD and C from Msg.X, Msg.CKD and Msg.C respectively using the function ZZFromBytes (ZZ& x, const unsigned char *p, long n)

3. Use X to calculate $R_i = X \bmod N_i$, where N_i is the prime number associated with the user.
4. Decrypt R_i with receiver's decryption key d_i to obtain the session decryption key d, using the RSA decryption function void Decrypt (ZZ c, ZZ d, ZZ n).
5. Decrypt CKD using d (obtained in the previous step), to find out if the receiver is an intended receiver (i.e., if d = Decrypt (CKD, d, n))
6. If the receiver is an intended receiver, use the session decryption key d to decrypt the message M from C. i.e., M = Decrypt (c, d, n)
   Else ignore the message.

3.2.3 Handling Leaves and Joins

All the leaves and joins can be dynamically handled in the proposed protocol. Whenever a new user joins, he will only be able to decrypt the messages that will be communicated after his join. Whenever an existing user leaves the group, he will not be able to decrypt any more messages that will be communicated in the group after he leaves. There can be situations where there are multiple leaves and joins at the same time. The following are the possible scenarios and solutions to handle simultaneous leaves and joins of members.

Recall that every user in the group is associated with an integer N_i and a public key (e_i, n_i). When there are n users, the lock is constructed from the following n congruences.

$$
\begin{aligned}
&X \equiv R_1 \bmod N_1 \\
&X \equiv R_2 \bmod N_2 \\
&\quad . \\
&\quad . \\
&X \equiv R_i \bmod N_i \qquad \text{for all users } u_i \\
&\qquad\qquad\qquad\quad \text{for all } R_i,\ R_i \le N_i \\
&X \equiv R_n \bmod N_n
\end{aligned}
\tag{4.1}
$$

The common solution X is then calculated from the following equation.

$$X = ((L/ N_1)* R_1*f_1 + (L/ N_2)* R_2*f_2 +. \ldots\ldots+(L/ N_i)* R_i*f_i + \ldots+(L/N_n)* R_n*f_n) \bmod L$$

Where

$$L = N_1* N_2*\ldots N_i*\ldots\ldots\ldots\ldots.* N_n \text{ and}$$

$$f_i *(L/N_i) \equiv 1 \pmod{N_i}$$

Whenever there is a change in membership, the only change that needs to be made is the construction of the lock X.

Single Join

When a new user $n+1$ joins with an associated integer N_{n+1} and public key (e_{n+1}, n_{n+1}), the new set of congruences from which the lock will be calculated, is the set of equations 4.1 plus $X \equiv R_{n+1} \bmod N_{n+1}$. So, the new set of equations will be:

$$X \equiv R_1 \bmod N_1$$

$$X \equiv R_2 \bmod N_2$$

.

$$X \equiv R_i \bmod N_i$$

.

$$X \equiv R_{n+1} \bmod N_{n+1}$$

The solution to the lock X is now calculated from the set of n+1 congruences.

$X = ((L'/ N_1)* R_1*f_1 + (L'/ N_2)* R_2*f_2 +. \ldots\ldots+(L'/ N_i)* R_i*f_i + \ldots+(L'/N_{n+1})* R_{n+1}*f_{n+1}) \bmod L'$ where

$$L' = L*N_{n+1} \quad \text{and}$$

$$f_i *(L'/N_i) \equiv 1 \pmod{N_i} \quad \text{where } 1 <= i <= n$$

Single Leave

When an existing user *i* leave the group, remove the corresponding congruous equation in finding the common solution X. i.e., the solution X will be:

$X = ((L'/ N_1)* R_1*f_1 + (L'/ N_2)* R_2*f_2 +. \ldots\ldots+(L'/ N_{i-1})* R_{i-1}*f_{i-1} + (L'/ N_{i+1})* R_{i+1}*f_{i+1} + \ldots+(L'/N_n)* R_n*f_n) \bmod L'$

where

$L' = L/N_i$ and

$f_j*(L'/N_j) \equiv 1 \pmod{N_j}$ where j ! =i and1 <= j <= n

Single Leave and Single Join

Let user *i* leave the group and user *n+1* join the group. The new lock will be:

$X = ((L'/ N_1)* R_1*f_1 + (L'/ N_2)* R_2*f_2 +. \ldots\ldots+(L'/ N_{i-1})* R_{i-1}*f_{i-1} + (L'/ N_{i+1})* R_{i+1}*f_{i+1} + \ldots+(L'/N_{n+1})* R_{n+1}*f_{n+1}) \bmod L'$

where

$L' = L* N_{n+1}/N_i$ and

$f_j*(L'/N_j) \equiv 1 \pmod{N_j}$ where j ! =i and1 <= j <= n+1

Single Leave and Multiple Joins

Let user *i* leave the group and the users *n+1*, *n+2*…n+*k* join the group. The new lock will be:

$X = ((L'/ N_1)* R_1*f_1 + (L'/ N_2)* R_2*f_2 +. \ldots\ldots+(L'/ N_{i-1})* R_{i-1}*f_{i-1} + (L'/ N_{i+1})* R_{i+1}*f_{i+1} + \ldots+(L'/N_{n+1})* R_{n+1}*f_{n+1} + \ldots.. + (L'/N_{n+k})* R_{n+k}*f_{n+k}) \bmod L'$

where

$L' = (L* N_{n+1}* N_{n+2}*\ldots\ldots..* N_{n+k})/N_i$ and

$f_j*(L'/N_j) \equiv 1 \pmod{N_j}$ where j ! =i and1 <= j <= n+k

Multiple Leaves and Single Join

Let the users *i, j,..,l* leave the group and user *n+1* join the group. The new lock will be:

$X = ((L'/ N_1)* R_1*f_1 + (L'/ N_2)* R_2*f_2 +. \ldots\ldots+(L'/ N_{i-1})* R_{i-1}*f_{i-1} + (L'/ N_m)* R_m*f_m + \ldots+(L'/N_{n+1})* R_{n+1}*f_{n+1}) \bmod L'$

where

$L' = (L* N_{n+1})/(N_i*\ldots\ldots..* N_l)$ and

$f_j*(L'/N_j) \equiv 1 \pmod{N_j}$ where j! =i…l and1 <= j <= n

Multiple Leaves and Multiple Joins

Multiple Leaves and Joins can be handled in the same way as Multiple Leaves and Single Join.

3.2.4 Handling large group of users

The problem with the secure lock protocol is that the Chinese Remainder Theorem (CRT) requires complex computation. The secure lock X has to be calculated from the following equations.

$X = ((L/ N_1)* R_1*f_1 + (L/ N_2)* R_2*f_2 +. \ldots\ldots+(L/ N_i)* R_i*f_i + \ldots+(L/N_n)* R_n*f_n) \bmod L$

Where

$L = N_1* N_2*\ldots Ni*\ldots\ldots\ldots\ldots* N_n$ and

$f_i*(L/N_i) \equiv 1 \pmod{N_i}$ where 1 <= i <= n

The time to compute the lock and the length of the lock (hence the transmission time) is proportional to the number of users in the system. This computation becomes really expensive when there are a large number of users. The complexity of the CRT algorithm is

dominated by the multiplication of the integers N_i in computing L. There are few solutions to this problem.

1. Use efficient multiplication algorithms. For example, use divide-and-conquer strategy. To multiply two n bit numbers X and Y, we can use the divide-and-conquer strategy as follows.

Let $X = a*2^{n/2} + b$

$Y = c*2^{n/2} + d$

$U = (a+b)*(c+d)$

$V = a*c$

$W = b*d$

Then

$$XY = V*2^n + (U-V-W)*2^{n/2} + W$$

2. Use parallel multiplication hardware and special CRT computing hardware (Processing unit of systolic array with block interleaving technique for multiplications and square operations).

3. Partition the users into a number of subgroups with suitable size. Then construct a lock for each subgroup using the Chinese Remainder Theorem. A lock is sent while a lock for the next subgroup is constructed, so that construction and transmission time of these locks can be overlapped.

All the above methods increase the speed to some extent at the expense of additional hardware and efficient multiplication algorithms. But still, the computational problems will be there if the number of users in the system is substantially large. Hence, we propose a protocol for dynamic secure multicasting with a large number of users. We call it **Distributed Secure Lock Protocol** (DSLP). This protocol is developed based on the secure lock protocol. This protocol can also make use of the above techniques to work faster.

Chapter 4: Distributed Secure Lock Protocol (DSLP)

The users in the system are divided into different groups based on their location (IP address). All the users from a local area network (LAN) are grouped together. Each group has a Group Manger (GM) and there is a Centralized Authority (CA) who has the tightest control over the group managers and the users in the multicast group. This approach is very suitable for the applications with high security demands. The Centralized Authority generates the keys. The DSLP is easy to implement and maintain and poses little load on the network and the users. User does not have to calculate the lock and transmit the lock as in the secure lock protocol (SLP). Diagrammatically the system looks like the following.

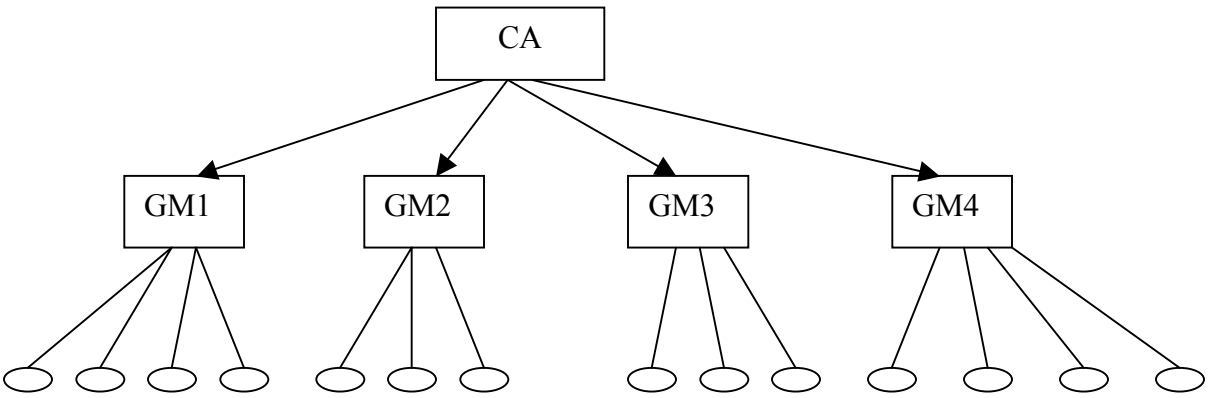


Figure 4.1: Proposed multicasting system based on distributed secure lock

The key management is not completely hierarchical as in centralized tree-based system, but there is a 2-level hierarchy. When a user joins the multicast group, he has to register with the centralized authority through the corresponding group manager by submitting the credentials (public key etc.) to the group manager. Once the user is authorized, the GM assigns him an integer, which can later be used as the identifier. Once the user successfully registers with the group, he can participate in the group communication.

4.1 Key Generation and Management

The centralized manager generates the session key (can be public or private) and distributes to the group managers using secure lock. Let the n group managers be denoted as g_1, g_2,……..g_n. Assume each member g_i in the group G has been assigned an integer ng_i. Let ng_1, $ng_{2,.....}$,ng_n be pair wise relatively prime; Let e_{gi} be g_i's public enciphering key and d_{gi} be g_i's secret deciphering key. Let *e* be the session encryption key and *d* be the corresponding private key. The CA has to send both the session encryption key and the decryption key to the group managers who in turn send the session keys to the group members. Assuming that public key cryptography is used, only the session decryption key has to be locked using Chinese Remainder Theorem which matches with the keys of the members of group G. The format of the sent-out message is given as follows:

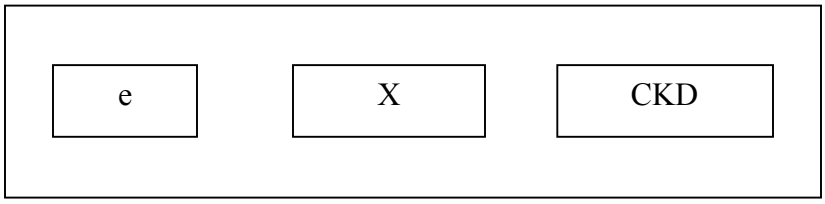


Figure 4.2: Message sent by the CA to the GMs

Where e is the session enciphering key

X is the lock, being the solution of the following congruences:

$X \equiv E\ e_{gi}(d) \bmod ng_i$, for all g_i in G

CKD is the cipher text of *d* which is enciphered by *e*; i.e., CKD = $E_e(d)$, and used by the receivers to check whether the CA wants to send him the key or not.

The following is the summary of what the CA does whenever he sends the keys out.

- Generate the session keys *e* and *d*.
- Use the CRT to compute the common solution X from the following congruences

$X \equiv Rg_1 \bmod ng_1$

……..

$X \equiv Rg_i \bmod ng_i$ for all g_i in G

……

$X \equiv Rg_n \bmod ng_n$

where Rg_i is the cipher text of d enciphered by e_{gi}; i.e., $Rg_i = E\ e_{gi}(d)$

- Compute the CKD.

  $CKD = E_e(d)$

- Build the data structure with the fields *e*, X and CKD, and transmit it to the group managers (multicast to the group managers group).

4.2 Group Managers' Role

Each group manager receives the following message structure from the CA.

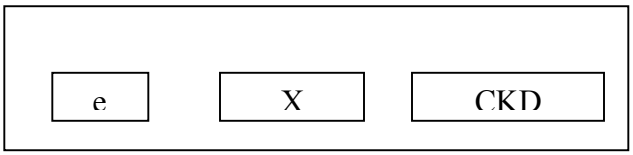


Figure 4.3: Message received by a GM from the CA

The group manager has to obtain the session deciphering key from X and send it to the group members by locking it using CRT. The following is the summary of what the GM does when he receives the new session key from the CA.

- Receives the message M (e, X, CKD)
- Compute d from X by using its deciphering key d_{gi}.

  i.e., to compute $d = Dd_{gi}\ (X \bmod ng_i)$

  $= Dd_{gi}\ (Dd_{gi})$

- Compute $D_d(CKD)$.
- If $D_d(CKD) = d$, then the group manager g_i is an intended receiver

  Else discard the message.

Now that the GM has obtained the session deciphering key d, he has to send it to the group members by locking it.

Let the x members in a group be denoted as u_1, u_2,……..,u_x. Assume each member u_i in the group g_i has been assigned an integer n_i. Let n_1, n_2,..... ,n_x be pair wise relatively prime; Let e_i be u_i's public enciphering key and d_i be u_i's secret deciphering key. The group manager has to send both the session encryption key *e* and the decryption key *d* to the group members. Assuming that public key cryptography is used, only the session decryption key has to be locked using the Chinese Remainder Theorem, which matches with the keys of the members of group G. The format of the sent-out message is as follows:

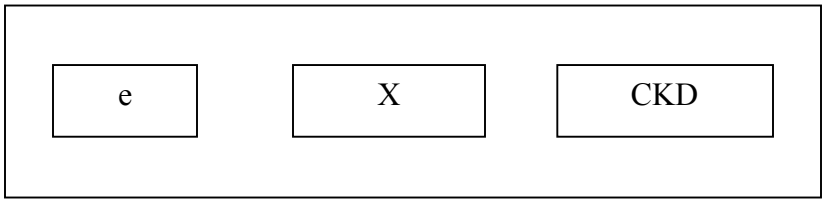


Figure 4.4: Message sent by the GM to its group members

where e is the session enciphering key received from the CA

X is the lock, being the solution of the following congruences:

$$X \equiv Ee_i(d) \bmod n_i, \text{ for all } u_i \text{ in } g_i$$

CKD is the cipher text of *d* which is enciphered by *e*; i.e., CKD = $E_e(d)$, and used by the receivers to check whether the GM wants to send him the key or not.

4.2.1 GM Algorithm

1. Receive the session keys *e* and *d*.
2. Use the CRT to compute the common solution X from the following congruences

$$X \equiv R_1 \bmod n_1$$

……..

$X \equiv R_i \bmod n_i$ for all u_i in g_i and $1 <= i <= x$

……

$X \equiv R_x \bmod n_x$

Where R_i is the cipher text of d enciphered by e_i; i.e., $R_i = E\ e_i(d)$

3. Compute the CKD.

   $CKD = E_e(d)$

4. Build the data structure with the fields *e*, X and CKD, and transmit it to the group members (multicast to the group members).

All the GMs will have to do is to send the keys to their respective group members whenever they receive a net set of keys from the CA.

4.3 Group Members

Once the members receive the session keys, they can encrypt the message with the session encryption key *e* and transmit it to the multicast group. There is no overhead associated in sending the message as in the original secure lock protocol, since the keys are not appended to the message. Hence, the transmission time will be less.

4.4 Handling Joins and Leaves

When a new member joins the multicast group or an existing member leaves the group, the CA generates new session keys and transmits them to the group managers (GMs) in a secure way as described in the previous sections. The GMs receive the keys and send them to their respective group members. Now, the members belonging to the current session can make use of the keys for communicating. The same procedure is followed when there are multiple joins and/or multiple leaves.

4.5 Performance Evaluation

The properties of the proposed Distributed Secure Lock Protocol (DSLP) will now be explored in terms of functionality, performance and how it deals with security threats. The following is the summary of the functionalities offered by DSLP.

1. Allows multiple leaves and joins
2. Supports exclusion of participants
3. Allows establishment of group-key to achieve privacy and/or authenticity
4. CA holds the group managers' keys
5. GMs hold the keys of their respective members
6. The computation of the lock is distributed across the GMs.
7. No trust in third parties is required
8. One central controlling authority provides tightest control over the participants
9. Controlling entity must know all participants but doesn't have to store the keys of the multicast group members
10. Easy to join and separate groups
11. Setup implosion is not an issue
12. There is a single point of failure, but easily recoverable as the GMs have their group and session members' information
13. Members do not need to store any keys other than the session key
14. There is no additional message attached to the original message

4.5.1 Usability

This distributed protocol is better suited for high-security applications and also fits into dynamic conferencing with a dedicated chair for all the sessions. The CA does not have to hold all the group members' keys but has to hold all the group members' IDs. This approach compares favorably with existing protocols in terms of simplicity, computing requirements and achieved security.

4.5.2 Performance

Resource usage is a critical point in all applications that offer cryptographic functions. Relevant costs for the CA and the GMs are:

- CPU consumption
- Memory consumption
- Communication bandwidth
- Typical end-to-end operational delay

The CA has to store

1. IDs of group members and the GMs
2. Public keys of the GMs

Usually these integers are as large as 512 bytes and even more than that. The GMs have to store the IDs and public keys of their respective group members. There will be temporary memory used by both the CA and the GMs while computing the lock. CPU consumption by the CA depends on the number of GMs present in the group, since the amount of computation done depends on the time and memory taken in computing the lock using the Chinese Remainder Theorem (CRT). The CA will have to do less computation compared to the computation done by the GMs. The GMs will have more members and hence there will be more memory and time taken in computing the lock using the CRT. The results will be presented later in the chapter 5.

Joins and leaves introduce some overhead on the network bandwidth. Each time a user joins or leaves, the CA has to create a session key and send it to the GMs who in turn send the key to the group members after locking the key. The transmission of the lock (the lock can be huge depending on the number of members of the group) will take up network bandwidth. The transmission of the message by the members does not introduce any overhead. There is no attachment to the message while transmitting it. Hence, the time to transmit the message depends on the encryption algorithm used, since the message that is transmitted is the encrypted text of the message. If the encryption algorithm produces much larger cipher text than the plain message, it takes more time to transmit the message.

4.6 Security Attacks

The security of the proposed DSLP is the same as that of the cryptography protocols used. In our protocol, for each GM g_i, the CA enciphers the session decryption key d with the group manager's encryption key e_{gi}, separately and mixes these cipher texts by the CRT algorithm. We have the following congruences

$$X \equiv Rg_1 \bmod ng_1$$

……..

$$X \equiv Rg_i \bmod ng_i \text{ for all } g_i \text{ in } G \quad \text{----------} \rightarrow (1)$$

……

$$X \equiv Rg_n \bmod ng_n$$

where Rg_i is the cipher text of d enciphered by e_{gi}; i.e., $Rg_i = Ee_{gi}(d)$--→ (2)

A cipher is breakable if it is possible to determine the plain text from a given cipher text. In general, any multicasting protocol is considered secure if it doesn't degrade the security achieved by the cipher. Ciphers are acceptable if they withstand any known-plaintext attacks. So, if the multicasting protocol with a cipher that withstands any known-plaintext attacks, can still withstand any known-plaintext attacks, then it is secure. An intruder can attack our protocol by the following means.

1. *Attack to find the deciphering session key d.*

   The intruder who doesn't belong to the group has to solve the equations (2) in order to get the key d. To do that, he has to break all the different ciphers in (2) encrypted with different keys e_{gi}, which are secure against known-plain text attacks. Hence, the protocol retains the security of the cipher.

2. *Attack to find the secret key.*

   If the intruder belongs to the group but not to the session, he will try to obtain the secret key of a session member. Even if he knows the (cipher, plain) pairs, the intruder cannot obtain more information from (2). Hence, the intruder will have same level of difficulty here as in case 1.

Chapter 5: Results and Conclusions

5.1 CRT Computation Results

As is stated in chapters 3 and 4 earlier, the main computational unit in the proposed protocol DSLP is the Chinese Remainder Theorem. The results are obtained by running the programs written in C++ using the library NTL on Linux Machines with P-II 400 MHZ processor. The following table shows the time taken to compute the CRT for different values of N where N is the size of the group. The chosen size for each N_i is 1024 bits and for each R_i is1000 bits. These results are obtained by executing the program using the NTL library.

N	Time in seconds
10	0
20	1
50	2
100	6
150	9
200	14
250	20
300	25
400	38
500	54
600	73
700	93
800	116
900	143
1000	170
1100	201

Table 5.1: Time taken by CRT for various values of N

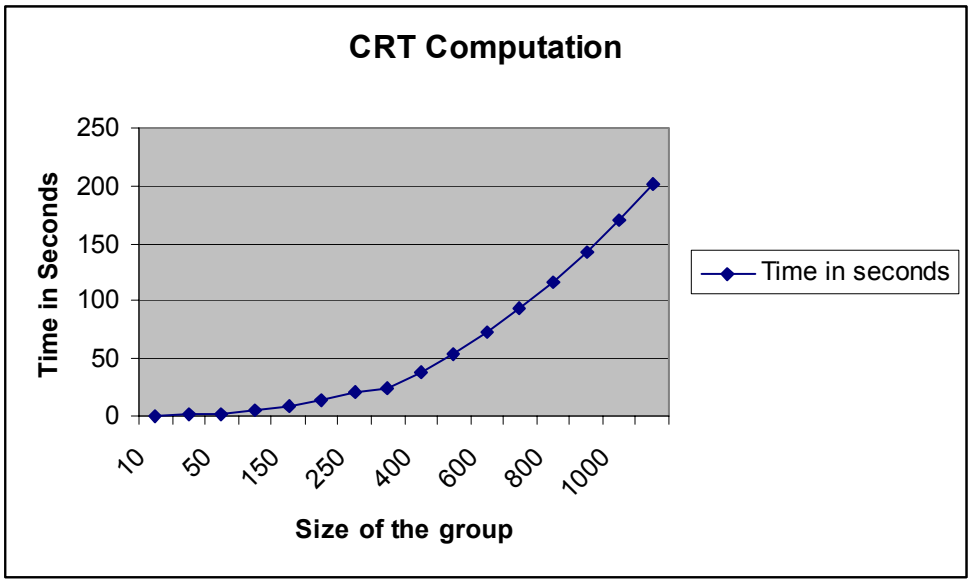


Figure 5.1: Time taken in seconds by CRT Vs Size of the group

It is clear from the above graph that as the size of the group increases the time taken to compute the CRT increases. When the size of the group is less than 150, the time to compute CRT is under 10 seconds, which is negligible when compared with the transmission time in most of the cases. In time critical applications, the amount of time taken to build the lock should be reduced as much as possible. As an example consider the case of 1000 users. If SLP is used, the amount of time taken to compute the lock itself is 170 seconds (from table 5.1). On the other hand, if DSLP is used with 10 group managers and 100 members in each group approximately, then the time taken by the CA to build the lock is close to zero and the time taken by each GM to build the lock for its group members is about 6 seconds. The only overhead in DSLP would be building and transmitting the lock twice, which should be far less than the CRT computation time incurred by SLP.

5.2 Decryption Results

The following table shows the decryption time taken by RSA and Rabin's public key algorithms for different sizes of the message. The size of the keys used for both the algorithms is 1024 bits. The results are obtained by running the algorithms using the NTL library.

Size	RSA	Rabin's
500	0	0
1000	1	0
1500	1	0
2000	1	0
4000	1	0
10000	3	1
20000	7	1
40000	14	2
50000	16	2
60000	19	3
80000	26	4
10000	41	5
12000	40	6

Table 5.2: Decryption times of RSA and Rabin's public key algorithms for different sizes of the message.

It is obvious from the above table that RSA takes a lot more time to decrypt a given message than the time taken by Rabin's public key algorithm. Both the algorithms take negligible amount of time to encrypt a message for any size of the message. Hence, RSA is normally used to encrypt the key and in turn the key is used to encrypt the message.

5.3 Length of the Lock

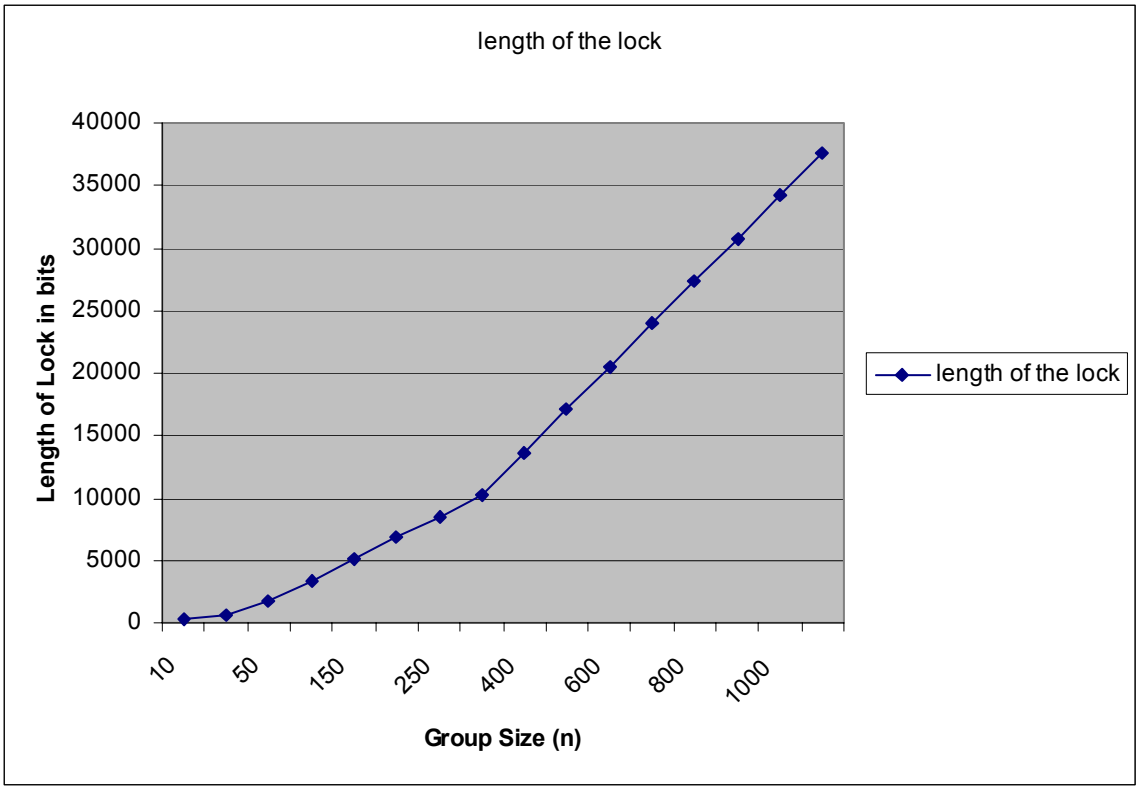


Fig 5.2: Length of the Lock in bits Vs Group Size (n)

The above graph shows the relationship between the length of the graph and the number of members in the group. It is clear that the length is proportional to the group size. Consider the case where the group size is 1000. The length of the lock is 34255 bits. If the size of the cipher text of the original message is, say, 340 bits. The size of the lock is nearly 1000 times more than the size of the original message itself, causing a lot of overhead on both the sender and the network. If SLP is used in a group of 1000 users, then the lock is 1000 times more than the original message (assuming the message is around 340 bits). On the other hand, if DSLP is used with 10 GMs and around 100 members in each group, then the size of the lock constructed by each GM will be approximately 3425 bits, 10 times less than that of SLP.

5.4 Conclusions

An increasing number of applications today require secure multicast services in order to restrict group membership and enforce accountability of group members. In the future, it is expected that the volume of secure multicasting data traffic will increase significantly. The major problems with most of the proposed protocols so far are the scalability of the key distribution, the complexity of computing the keys, the complexity of re-keying and the addressing of dynamic groups. Some of these protocols also require the members of the groups to hold a large number of keys causing overhead on the participating users. In large groups, whenever there is a change in the group membership (existing members leaving the group, new members joining the group), key changes that are required also pose a major problem. As the frequency of group membership change increases it is necessary to reduce the cost of the key distribution process.

In this thesis, the secure broadcasting protocol of Chiou and Chen has been tuned to multicasting and ways to handle the joins and leaves of the multicasting group are discussed. The proposed protocol DSLP, which is based on the idea of Chinese Remainder Theorem and one-level hierarchy, has been implemented in a LAN and tested for a relatively small group of users. The results of the CRT computation have been obtained for groups of size varying from 10 – 1100. The time taken to compute the decryption using the RSA and the Rabin's Public Key algorithms have been obtained for encrypted messages of size varying from 500 bits to 12000 bits. RSA takes relatively a longer time to decrypt a message compared to the time taken by Rabin's algorithm. Hence, RSA is normally used to encrypt the key and in turn the key is used to encrypt the message.

The proposed protocol DSLP expected to work better than the SLP due to the following reasons:

1. DSLP works as efficiently as SLP for small groups. But, as the size of the group increases, DSLP works faster than SLP.

2. Members will no longer have to compute the lock in DSLP. The only overhead on the users is unlocking the lock sent by their group managers, encrypting the message before sending and decrypting the message received.
3. The overhead of computing the lock is shifted to the group managers and the centralized authority in DSLP. That implies that the CA keeps track of the session members. Whenever there is a change in the session group, the CA will create a new session key and send it to the GMs. In SLP, the senders have to contact the group manager each time, to know the current session members and to construct the lock. Otherwise, all the members have to keep track of the session members, which would be an overhead again.
4. Since, there are no attachments to the messages exchanged in the group, there is no additional overhead on the network bandwidth. In SLP, the lock is always attached to the message and sometimes the lock can be bigger than the message itself when there are a large number of users in the multicast group. Hence, there is overhead on the transmission of the message in SLP.
5. For the same reason, the transmission time will be much less in DSLP than in SLP.
6. In SLP, whenever a sender sends a message, he has to construct a new key and the message has to be encrypted with the new session key. This is done even when the group is static i.e., even if there are no leaves and joins in the group; a new key is constructed whenever a message is exchanged in the group.
7. The computation of the lock is centrally distributed in DSLP. The CA and the GMs, share the responsibility. Hence, no single person will have to compute the lock for all the members in the group.
8. DSLP can make use of all the efficient algorithms and CRT computation techniques that are adopted in SLP.

9. Both have the same level of security. The security depends on the security of the cryptographic algorithms employed by the protocols.
10. One disadvantage of DSLP over SLP is that there is a single point of failure. If the CS goes down, the whole system will go down. Hence, there needs to be a back up mechanism (which is not very hard, as the GMs have the necessary information about their group members).
11. Another disadvantage of DSLP is that in SLP there is no centralized authority as in DSLP. Hence, SLP is suitable for all kinds of applications whereas DSLP is better suited to applications where the tightest security is required.

In future, work can be done in developing ways to withstand the single point of failure at the CA and failures at the group managers. DSLP can be implemented in different networks and tested for a large group of users.

References

[1] A. S. Tanenbaum, Computer Networks, Englewood Cliffs, NJ, Prentice-Hall, 1996.

[2] J. Snoeyink, S. Suri, G. Varghese, A Lower Bound for Multicast Key Distribution, IEEE INFOCOM 2001.

[3] Guang-Huei Chiou, Wen-Tsuen Chen, Secure Broadcasting using secure lock, IEEE Transactions on software engineering, vol.15, No.8, pp. 929-934, August 1989.

[4] B. Schneier, Applied Cryptography, John Wiley & Sons, Inc., 2001.

[5] A. Salomaa, Public-Key Cryptography, Springer, Heidelberg, 1996.

[6] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key management for secure Internet multicast using boolean function minimization techniques. In Proceedings of IEEE INFOCOM, New York, March 1999.

[7] D. E. Comer, D. L. Stevens, Internetworking with TCP/IP, Englewood Cliffs, NJ, Prentice-Hall, 2000.

[8] Suvo Mittra, Iolus: A Framework for Scalable Secure Multicasting, Proceedings of ACM SIGCOMM'97, Cannes, France, pp. 277-288, 1997.

[9] W. R. Stevens, Unix Network Programming, Englewood Cliffs, NJ, Prentice-Hall, 1999.

[10] K. Tan, H. Zhu. “A conference key distribution scheme based on the theory of quadratic residues”, Computer Communications, Vol. 22, No. 8, May 1999, 735-738.

[11] G. Caronni, M. Waldvogel, D. Sun, and B. Plattner. Efficient security for large and dynamic groups. Technical Report TIK Technical Report No. 41, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, February 1998.

[12] K Wu, S Ruan, F Lai, C Tseng, On Key Distribution in Secure Multicasting, IEEE 2000.

[13] D.E Denning, Cryptography and Data Security, Addison – Wesley, Reading, MA, 1982.

[14] L. R. Dondeti, S. Mukherjee, and A. Samal. A Dual Encryption Protocol for Scalable Secure Multicasting. In Proceedings of the Fourth IEEE Symposium on Computers and Communications, Red Sea, Egypt, July 1999.

[15] NTL: A Library for doing Number Theory, http://shoup.net/ntl/

[16] Multicast programming, http://www.tldp.org/HOWTO/Multicast-HOWTO-6.html

[17] T. Hardjono and G. Tsudik. IP multicast security: issues and directions. November 2000.

[18] M. Steiner, G. Tsudik, and M.Waidner, "Diffie-hellman key distribution extended to group communication," in *Proc. 3rd ACM Conf. Computer and Communications Security*, Mar. 1996.

[19] K.C. Chan and S.H. Chan, "Distributed Servers Approach for Large-Scale Secure Multicast," *IEEE Journal on Selected Areas in Communications* special issue on *Network Support for Multicast Communications,* Vol. 20, No. 8, pp. 1500-1510, October 2002.

[20] H. Chu, L. Qiao, and K. Nahrstedt. A secure multicast prorocol with copyright protection. In Proceedings of IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology, San Jose, CA, January 1999.

[21] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The versakey framework: Versatile group key management. IEEE Journal on Selected Areas in Communications(special Issue on Middleware), 17(9):1614-1631, September 1999.

Appendix A: NTL

NTL is a high-performance, portable C++ library providing data structures and algorithms for

i. Arbitrary length integers;
ii. Vectors, matrices, and polynomials over the integers and over finite fields; and
iii. Arbitrary precision floating point arithmetic.

NTL provides high quality implementations of state-of-the-art algorithms for:

a. Arbitrary length integer arithmetic and arbitrary precision floating point arithmetic
b. Polynomial arithmetic over the integers and finite fields including basic arithmetic, polynomial factorization, irreducibility testing, computation of minimal polynomials, traces, norms, and more
c. Lattice basis reduction
d. Basic linear algebra over the integers, finite fields, and arbitrary precision floating point numbers

NTL is written and maintained by Victor Shoup [] with some contributions made by others. The implementation of the encryption and decryption algorithms, generation of random prime numbers and the arithmetic operations on the large integers required the deployment of NTL. NTL can generate integers of size more than thousands of bytes and can perform arithmetic operations on them.

The integers are declared with the keyword ZZ. For example an integer x is declared as ZZ x. All the basic operations +, -, *, / and % can be carried out in the way they are done in C, C++ or any other high-level language. Vectors are available in NTL as a replacement for arrays. The size of the vector can be changed dynamically. Vectors are used in the following way.

```
vec_ZZ v;       //declare a vector
```

v.SetLength(n); //set the size of the vector v

v.length(); //returns the size of the vector v

The following are some of the other operations that NTL can support. These operations are used to implement secure multicasting.

**/

Modular Arithmetic

The following routines perform arithmetic mod n, where n > 1.

All arguments (other than exponents) are assumed to be in the range 0...n-1. Some routines may check this and raise an error if this does not hold. Others may not, and the behavior is unpredictable in this case.

**/

void SqrMod (ZZ& x, const ZZ& a, const ZZ& n)

ZZ SqrMod (const ZZ& a, const ZZ& n) where x = a^2 % n

void InvMod (ZZ& x, const ZZ& a, const ZZ& n)

ZZ InvMod (const ZZ& a, const ZZ& n) where x = a^{-1} mod n (0 <= x < n); error occurs if inverse is not defined

void PowerMod (ZZ& x, const ZZ& a, const ZZ& e, const ZZ& n)

ZZ PowerMod (const ZZ& a, const ZZ& e, const ZZ& n) where x = a^e % n

void PowerMod (ZZ& x, const ZZ& a, long e, const ZZ& n)

ZZ PowerMod (const ZZ& a, long e, const ZZ& n) where x = a^e % n

The following functions are used to do conversion between integers and byte sequences. The conversion is necessary to send the data over the socket while communicating in the multicast group.

void ZZFromBytes (ZZ& x, const unsigned char *p, long n)
ZZ ZZFromBytes (const unsigned char *p, long n) where x = sum (p[i]*256^i, i=0..n-1).
NOTE: in the unusual event that a char is more than 8 bits, only the low order 8 bits of p[i] are used.

void BytesFromZZ (unsigned char *p, const ZZ& a, long n) computes p[0..n-1] such that abs(a) == sum(p[i]*256^i, i=0..n-1) mod 256^n.

long NumBytes (const ZZ& a) and long NumBytes (long a) both return number of base 256 digits needed to represent abs (a). NumBytes (0) == 0.

The following code shows the RSA encryption and decryption functions implemented using the NTL library.

```
ZZ Encrypt (ZZ m, ZZ n, ZZ e){

      ZZ c;
      c = PowerMod (m, e, n);
      cout << "\n The Cipher Text is" << c << "\n";
      return c;
}
ZZ Decrypt (ZZ c, ZZ d, ZZ n){
      cout << "Decrypting\n";
     return PowerMod (c, d, n) << "\n";
}
```

Appendix B: Unix Multicast Programming

There are two system calls that are used to pass information to the kernel and retrieve the multicast information from the Kernel. They are **setsockopt ()** and **getsockopt ()** and their prototypes are:

int getsockopt (int *s*, int *level*, int *optname*, void* *optval*, int* *optlen*);

int setsockopt (int *s*, int *level*, int *optname*, const void* *optval*, int *optlen*);

The parameter *s* is the socket of the family AF_INET and is of the type SOCK_DGRAM or SOCK_RAW. The second parameter, *level,* identifies the layer, which handles the option and is always IPPROTO_IP for multicast programming. `optname` identifies the option we are setting/getting. Its value (either supplied by the program or returned by the kernel) is `optval`. The optnames involved in multicast programming are the following:

	setsockopt ()	getsockopt ()
IP_MULTICAST_LOOP	yes	yes
IP_MULTICAST_TTL	yes	yes
IP_MULTICAST_IF	yes	yes
IP_ADD_MEMBERSHIP	yes	no
IP_DROP_MEMBERSHIP	yes	no

`optlen` carries the size of the data structure `optval` points to. Both `setsockopt()` and `getsockopt()` return 0 on success and -1 on error.

IP_MULTICAST_LOOP

If the sender wants to receive the data back that he sent, then set the optname as IP_MULTICAST_LOOP and its value *optval* to 1. If no loop back is required set the value to 0.

IP_MULTICAST_TTL

If nothing is specified, then the default value will be set to 1 so that the message will be prevented from being forwarded beyond the Local Area Network. But, the time to live (TTL) parameter can be set to any value from 0 to 255.

```
u_char ttl;
    setsockopt (socket, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof (ttl));
```

IP_MULTICAST_IF

Using this parameter, the programmer can specify the interface that the multicast datagrams should be sent from.

IP_ADD_MEMBERSHIP

In order to join a group, a process has to inform its kernel that it is interested in joining a particular group. To do this, the process has to fill an ip_mreq structure, which is passed to the kernel. The ip_mreq structure has the following members.

```
struct ip_mreq {
        struct in_addr imr_multiaddress; /* IP multicast address of a group */
        struct in_addr imr_interface;    /* Local IP address of interface */ }
```

The first member imr_multiaddress is the address of a multicast group in which the process is interested. The second member imr_interface is the local IP address of the interface. This field can be set to INADDR_ANY so that the kernel will choose the interface. Once the ip_mreq structure is filled, the process has to call the setsockopt () system call as follows:

```
struct  ip_mreq mreq;
setsockopt (socket, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof (mreq));
```

IP_DROP_MEMBERSHIP

To leave a group, the process has to call setsockopt () with the *optname* set to IP_DROP_MEMBERSHIP. The system call will be called in the following way.

```
struct  ip_mreq mreq;
setsockopt (socket, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof (mreq));
```

Appendix C: Software

The following code is written in C++ on Linux using the NTL. To use the software, NTL needs to be deployed and configured. The instructions to download and configure the library can be found at http://shoup.net/ntl/.

/* multicast.c

The following program sends or receives multicast packets. If invoked with one argument, it sends a packet containing the data to an arbitrarily chosen multicast group and UDP port. If invoked with no arguments, it receives and prints these packets. Start it as a sender on just one host and as a receiver on all the other hosts Encryption scheme: RSA */

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <stdio.h>
#include <NTL/ZZ.h>
#include <NTL/vec_ZZ.h>
#include <unistd.h>
#include <stdlib.h>

#define EXAMPLE_PORT 6000
#define EXAMPLE_GROUP "239.0.0.1"
#define N 3
#define MSIZE 1000

ZZ ExtEuclid(ZZ, ZZ);
ZZ Encrypt(ZZ, ZZ, ZZ);
void Decrypt(ZZ, ZZ, ZZ);
ZZ crt();//CRT function

ZZ L;
vec_ZZ R, userid, pub_key1, pub_key2;

typedef struct message{
        unsigned char X[MSIZE], CKD[MSIZE], C[MSIZE], Pubkey[MSIZE];
};
```

```
int main(int argc)
{
  struct sockaddr_in addr;
  int addrlen, sock, cnt;
  struct ip_mreq mreq;

  message msg;

  ZZ p, q, n, m, c, CKD, X, e, Phi, d, counter, received, cipher, dd;
  long l, err;


  l = 500, err = 80;
  unsigned char text[MSIZE];
  unsigned char testmsg[MSIZE];

  p = GenPrime_ZZ(l, err);
  q = GenPrime_ZZ(l, err);

  R.SetLength(N);
  userid.SetLength(N);
  pub_key1.SetLength(N);
  pub_key2.SetLength(N);

 //the following values are to be read from the file, they were hard coded for testing
//purposes
  userid[i],  pub_key1[i] pub_key2[i] ;
  L = to_ZZ(1);
  dd // session decryption key
  n = p*q;
  Phi = (p-1)*(q-1);
  for(counter=2;counter<Phi;counter++){
        if(GCD(counter,Phi)==1){
              e = counter; break}}

  d = ExtEuclid(e, Phi);
  cout << "Enter the Message" << "\n";
  cin >> m; //input the plain text;
  cipher = Encrypt(n, e, m); //encrypt the plain text

  R[0] = Encrypt(pub_key1[0], pub_key2[0], d);
  R[1] = Encrypt(pub_key1[1], pub_key2[1], d);
  R[2] = Encrypt(pub_key1[2], pub_key2[2], d);

  CKD = Encrypt(n, e, d);//encrypt the decryption key
```

```
X = crt();

BytesFromZZ(msg.C, cipher, MSIZE);
BytesFromZZ(msg.CKD, CKD,MSIZE);
BytesFromZZ(msg.X, X, MSIZE);
BytesFromZZ(msg.Pubkey, n, MSIZE);
/* set up socket */
sock = socket(AF_INET, SOCK_DGRAM, 0);

if (sock < 0) {
  perror("socket");
  exit(1);
}
bzero((char *)&addr, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(EXAMPLE_PORT);
addrlen = sizeof(addr);

if (argc > 1) {
   /* send */
   addr.sin_addr.s_addr = inet_addr(EXAMPLE_GROUP);
    while (1) {
        time_t t = time(0);

        cout << "Enter any key  to transfer the Data.\n";
        cin >> c;

        cnt = sendto(sock, &msg, sizeof(msg), 0,
                    (struct sockaddr *) &addr, addrlen);
        if (c=='X'){
                cout << "Closing the session now..BYE" <<"\n";
                exit(1);
        }
        if (cnt < 0) {
          perror("sendto");
          exit(1);
        }
        else
                 cout << "Transmitted Successfully..Enter anykey to continue\n";
        cin >> c;

   }
} else {

   /* receive */
```

```
if (bind(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
  perror("bind");
     exit(1);
}
mreq.imr_multiaddr.s_addr = inet_addr(EXAMPLE_GROUP);
mreq.imr_interface.s_addr = htonl(INADDR_ANY);
if (setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
                  &mreq, sizeof(mreq)) < 0) {
     perror("setsockopt mreq");
     exit(1);
}
while (1) {

     cnt = recvfrom(sock, &msg, sizeof(msg), 0, (struct sockaddr *)&addr,
    (socklen_t*) &addrlen);

     ZZFromBytes(c, msg.C, MSIZE);
     ZZFromBytes(X, msg.X, MSIZE);
     ZZFromBytes(CKD, msg.CKD, MSIZE);
     ZZFromBytes(SessionPubkey, msg.Pubkey, MSIZE);

     R = X%userid[0];

     cout << "R=" << R << "\n";
     cout << "userid = " << userid[0] << "\n";

     cout << "Received C=" << c << "\n";
     cout << "Decrypted R=" << R << "\n";

     SessionDecryption = Decrypt(R%pub_key1[0], dd, pub_key1[0]);
     CKD = Decrypt(CKD%SessionPubkey, SessionDecryption, SessionPubkey);

     err1 = SessionDecryption - CKD;

     if(err1 == 0){
          cout << "I am a valid user\n";
          cout << "I have obtained:" << Decrypt(c%SessionPubkey,
          SessionDecryption, SessionPubkey) << "\n";
     }
     else
          cout << "I am trying to cheat";

     if (cnt < 0) {
       perror("recvfrom");
       exit(1);
     }
```

```
        else if (cnt == 0) {
                        break;
        }
  }}}//end of main



ZZ ExtEuclid(ZZ e, ZZ Phi){
        vec_ZZ g, u, v;
        ZZ x, y;
        long i=1;

        g.SetLength(MSIZE);
        u.SetLength(MSIZE);
        v.SetLength(MSIZE);

        g[0] = Phi;
        g[1] = e;
        u[0] = 1;
        v[0] = 0;
        u[1] = 0;
        v[1] = 1;

        while(g[i] !=0){
                g[i] = u[i]*Phi + v[i]*e;
                y = g[i-1]/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;
        }

        x = v[i-1];
      if(x>=0)
                return x;
        else
                return Phi + x;
}

ZZ Encrypt(ZZ n, ZZ e, ZZ m){

        ZZ c;
        //cout << "In Encryption" <<"m=" << m << "\ne=" << e << "\nn=" << n << "\n";
        c = PowerMod(m, e, n);
        //cout << "In Encryption\n";
        //cout << "\n The Cipher Text is" << c << "\n";
```

```
	return c;
}

void Decrypt(ZZ c, ZZ d, ZZ n){
	cout << "\n D=" << PowerMod(c, d, n) << "\n";
}


ZZ crt(){
	int i, j;
	ZZ X, p, x1, x2, x3;
	vec_ZZ y;
	y.SetLength(N);
	X =0, p=1, x3 = 0;
	int temp;
	int strength = 0;
	int SessionIds[N];

	printf("Enter the IDs of the users in the current session followed by -1\n");

	i = 0;
	cin >> temp;

	while (temp != -1) {
		SessionIds[i++] = temp;
		cin >> temp;
	}

	strength = i;
	for(i=0;i < strength; i++){
		temp = SessionIds[i];
		cout << "userid:" << userid[temp] << "\n";;
		L = L*userid[temp];
	}


	for(i=0; i < strength; i++){
		temp = SessionIds[i];

		p = to_ZZ(L/userid[temp]);
		x2 = userid[temp] - 2;
		y[i] = 1;

		y[i] = PowerMod(p%userid[temp], x2, userid[temp]);
		y[i] = y[i]%userid[temp];
	}
```

```
	for(i=0; i< strength; i++){
		temp = SessionIds[i];
		X = X + ((L/userid[temp])*R[temp]*y[i]);
	}

	X =X%L;
	return X;
}

/*
The following program implements the encryption and decryption methods of Rabin's
public key algorithm
*/


#include <NTL/ZZ.h>
#include <NTL/vec_ZZ.h>
#define N 60
ZZ Encrypt(ZZ, ZZ);

void Decrypt(ZZ, ZZ, ZZ, ZZ);

int main(){
	ZZ p, q, n, e, Phi, d, counter;
	vec_ZZ m,c;
	long l,err, l1;
	time_t t1, t2;
	int i;

	l = 1024, err = 80;
	l1 = 2000;
	m.SetLength(N);
	c.SetLength(N);

	do{
		p = GenPrime_ZZ(l, err);
		q = GenPrime_ZZ(l, err);
	} while(p%4 != 3 || q%4 != 3);

	for(i=0;i<N;i++)
		m[i] = GenPrime_ZZ(l1, err);

	n = p*q;

	cout << "\n m= " << m[N-1] << "\n";

	time(&t1);
```

```
        for(i=0;i<N;i++)
                c[i] = Encrypt(m[i], n);
        time(&t2);
        cout << "\n Time to Encrypt=" << (t2-t1) << "\n";

        time(&t1);
        for(i=0;i<N;i++)
                Decrypt(p, q, n, c[i]);
        time(&t2);
        cout << "\n Time to Decrypt = " << (t2-t1) << "\n";
}

ZZ Encrypt(ZZ m, ZZ n){
        ZZ c;
        c = PowerMod(m, 2, n);
        cout << "\n The Cipher Text is" << c << "\n";
        return c;
}

void Decrypt(ZZ p, ZZ q, ZZ n, ZZ c){

        ZZ a, b, gcd, r, s, x, y;
        XGCD(gcd, a, b, p, q);
        r = PowerMod(c%p, (p+1)/4, p);
        s = PowerMod(c%q, (q+1)/4, q);
        x = (a*p*s + b*q*r) % n;
        y = (a*p*s - b*q*r) % n;
        cout << "\nThe deciphers are \n" <<  x << "\n" <<  (-x%n) << "\n" << y << "\n"
        <<  (-y%n) <<"\n";
}
```

Vita

Phone: 540-818-3125
Email: koneni@vt.edu

Objective

To obtain a full-time position as a Software Developer

Education

M. S. **Computer Science & Applications (expected graduation: May 2003), GPA 3.8/4.0**
Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, VA, USA

M. Sc (Hons.) Economics & **B. E (Hons.)** Computer Science, June 2001, GPA 3.8/4.0
Birla Institute of Technology & Science (BITS, Pilani), INDIA (School Rank: 3 in INDIA)

Relevant Courses

Theory of Algorithms
Operating Systems
Computer Networks & Architecture
Microprocessors
Data Processing
Network Architecture & Programming
Report Writing
Principles of Management
Information Storage & Retrieval
Discrete Mathematics Operations Research
Database Management Systems
Numerical Analysis & Software
Internet Software
Network performance & Management
Grid Computing
Advanced Computer Organization

Experience

Software Engineer, Siri Technologies, Bangalore, India, *January 2001 – June 2001*
"Computer Based Training" using ASP, XML, Delphi.
Intern, National Institute of Oceanography, Goa, India, *May 1998 - July 1998*
"Geophysical 2D Modeling" using Borland C++.
Web Developer, Virginia Tech Linguistics Department, VA, *May 2002 – August 2002*
"Design and develop the website for the linguistics department at Virginia Tech"

Graduate Teaching Assistant, Virginia Tech Computer Science Department, VA, *Aug 2002 – May 2003*

Grading and helping students in the courses '**Operating Systems**', '**Data and Algorithm Analysis**' and '**Programming in C++**'.

Thesis Key Management Techniques for Dynamic Secure Multicasting for a large group of users

Relevant Projects

- Implementation of an Adaptable Reliable Transport Control Protocol over UDP
- Design and development OAI High Performance Search Engine
- Implementation of SNAPSTER: A generic file sharing system
- Implementation of a Concurrent Railway Reservation System
- Simulation of Job Scheduler and Process Scheduler
- Peer to Peer networking using RPC
- Study and implementation up to task switching of a 32-bit operating system
- Implementation of Inventory Management System
- Implementation of web based e-commerce shopping system

Skills

Languages	C, C++, Java, VB, Delphi, COBOL, Pascal and Fortran.
Operating Systems	Windows 2K,NT; Linux, Solaris, UNIX and Mac OS.
Networking	TCP/IP stack, related protocols and UNIX Socket Programming, HTTP, SMTP, FTP and SSL.
Other	SQL, XML, ASP, JSP, EJB, Servlets, JavaScript, Perl, IIS, Shell Script, RDBMS, ORACLE, OPNET, MATLAB, LAPACK, ELLPACK, Microsoft Front Page and Microsoft Access.

Scholastic Honors/Activities

- Joint Secretary, Economics-Finance Exhibition, 1998-99 APOGEE, BITS, Pilani, India Recipient of Scholarship (Tuition Fee Waiver) at undergraduate level
- GRE **2290**/2400 (Analytical: **800**/800, Quantative **800**/800)

VISA Status **F1**