RSA-BASED KEY COMPROMISED RESISTANT PROTOCOL (KCR) FOR LARGE DATABASES

A Thesis

Presented to

The Faculty of the Department of Computer Science

California State University, Los Angeles

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

By

Fatemah Mordhi Alharbi

December 2013

The thesis of Fatemah Mordhi Alharbi is approved.

Chengyu Sun

Huiping Guo, Committee Chair

Raj S. Pamula, Department Chair

California State University, Los Angeles

December 2013

ABSTRACT

RSA-based Key Compromised Resistant Protocol (KCR) for Large Databases

By

Fatemah Mordhi Alharbi

Data communication and networking are essential in our daily lives. Companies rely on computer networks and internetworks to exchange information, that is normally stored in large databases, with their customers. Nonetheless, it is not guaranteed that all networks are reliable; therefore, database content should be protected against any unauthorized access. One of the most powerful strategies that has been used in existing database security systems to protect databases is database encryption. Few of these systems are practical to be used with large databases since one important issue is not satisfactorily addressed which is concerning key management. It may take days to encrypt the huge databases. Imagine how the situation will be when a key is compromised! Straightforward solutions to address this problem demand that the keys used in database encryption to be replaced with new keys. Consequently, a database re-encryption process has to be executed. In this thesis, I propose "RSA-based Key Compromised Resistant Protocol (KCR)" to effectively address this problem. Experimental results show that the proposed protocol is sound.

ACKNOWLEDGMENTS

The most important contribution to the development of my project and thesis comes from Dr. Huiping Guo, my thesis advisor. I cannot express my appreciation in words for her magnificent cooperation along the way. Special thanks go to the staff and faculty of the department of Computer Science who gave me help whenever I needed it.

I have the honor to gratefully thank the Saudi Arabian government for granting me a scholarship to pursue my graduate studies in the United States of America. I thank the sincere employees of the Saudi Arabian Cultural Mission (SACM) who assisted me during my stay academically and socially.

I also would like to thank my beloved family for their encouragement and believing in their daughter. My final thanks go to my husband, Azzam, for his love and patience. He makes me believe that nothing is impossible if you are brave enough to face all obstacles.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1

Introduction

In the past, people had to exchange information physically back and forth, but nowadays we live in electronically connected world age. The invention of the Internet connects computer networks to serve people all over the world. The enormous amount of information resources available and ease of communication have made the Internet the most valuable tool in various settings of a person's life. However, the Internet is not considered a trustworthy network and the transmitted information is vulnerable to attacks which brings up the important issues of information security. Information security is the practice of protecting information and information systems against unauthorized access or alteration of information. It is a combination of data security and network security. Regarding data security, three security goals need to be achieved: confidentiality (hide data from attackers), integrity (keep data safe from unauthorized modifications), and availability (give authorized users access to their data) [1]. Along with these goals, there are five security services related to networking security: confidentiality, integrity, authentication, nonrepudiation, and authentication. Some of these services are for entities and messages sent or received by those entities [2]. An optimal security application should efficiently implement these goals and services.

Other than information security, facilitating the access to information resources is a fundamental subject to make people's life much easier. Databases are used as storages to gather information and to effectively store a large number of records. While data should be available to legitimate users, database administrators are responsible to restrict the access privileges for protection against any kind of attacks. According to [3] and [4], such approach is one of the strategies of database security. Other strategies and methods have been studied for some time. Early work [5], [6] considers physical security, operating system security, Database Management System (DBMS) security and data encryption as the major strategies that support database security. Regarding physical security, this strategy excludes any remote access without legal permission to the database. Although making physical contact with database contents is an expensive measure, it partially ensures data integrity [5],[7]. On the subject of operating system security, according to [8], any operating system protection model is consisted of three elements: objects within the system, entities who access these objects, and regulations that manage how entities access objects. However, it is difficult to govern entities from disclosing these regulations to unwanted parties; this type of attack is called client colluding attack. The third strategy concerns DBMS security. Available DBMS security solutions assume the database grants the appropriate privileges to legitimate users and this obviously is not guaranteed against Trojan horse attacks [9]. All of these strategies by themselves do not completely satisfy the requirements of database security [5], [6]. The forth strategy that uses encryption as a technology is the practical solution to dominate database security [5], [6], [7], [8]. Many systems have been developed and evaluated to support database encryption. For example, the scheme in [10] proposes an encryption mechanism to provide security and improve query processing efficiency. Many other schemes have been studied to improve some encryption algorithms for various purposes. For example, privacy protection using the Chinese Remainder Theorem [11] has been examined (e.g. [5], [6]). The studies are mainly based on encryption and anonymizing. Regarding key management for encipherment systems, strategies such as key division ciphers and commutative ciphers (e.g. [12], [13], [14], [15], [16], [17]) have been implemented.

Although these techniques and methods can improve the encryption and decryption procedures, none of them provides a key management system that prevents the re-encryption process of large databases when a key is compromised. Let me give an example to illustrate the problem. Consider a massive organization - industrial, government, or military - where all employees (referred to as users) are granted the appropriate privileges to access the organization's database that contains thousands of records. We assume that a modern Public Key Infrastructure (PKI) is available and all users are capable to perform their requests by applying the appropriate encryption techniques. Now suppose that a key pair (public key and private key) is generated and that the public key is used to encrypt the huge database and the private key, which is shared among all users in the organization, is used for decryption. Presume that an unauthorized user breaks the organization's security firewall and gains access to the decryption key. All database contents would be disclosed. One possible approach is to periodically change the database encryption and decryption keys, but this strategy addresses two main drawbacks. First, the overhead caused by frequently changing the keys is overkill. The second drawback is the re-encryption process of large databases that requires time and a lot of resources. To the best of my knowledge, there is no efficient and practical solution to such a problem.

In this thesis, I propose a novel key management protocol called "RSA-based Key Compromised Resistant protocol", referred to as KCR. The primary objective of this protocol is to effectively reduce the possibility re-encrypting large databases in case their keys have been compromised. KCR protocol consists of a set of strategies where the protocol operates practically. The protocol has been implemented and some experiments have been conducted for the purposes of evaluation. Simulation results show that my proposed protocol is works well.

The remainder of this thesis is organized as follows. Chapter 2 reviews some security cryptosystems and some details about databases. I describe KCR protocol in chapter 3, its implementation in chapter 4. I make concluding remarks in chapter 5.

CHAPTER 2

Overview of Cryptography and Databases

This chapter presents a brief description on the science of cryptography and some background about databases. By explaining these terminologies, the reader is prepared for the next few chapters, which discuss the KCR Protocol and its implementation.

Cryptography

Cryptography is the art of science of modifying information secretly in order to keep it hidden from attackers. Any cryptographic algorithm is based on mathematics to encrypt and decrypt data. In the old ages, algorithms rely on the secrecy of the encryption algorithm itself, yet their security needs are not adequate for real-world needs. For this reason, all modern algorithms move their security interests toward encryption and decryption keys; an encrypted message is decrypted if and only if the decryption key is mathematically related to the encryption key. These encipherment systems are classified into two categories: symmetric-key (secret-key) cryptography and asymmetric-key (public-key) cryptography. Asymmetric-key cryptography involves many mathematical computations for encryption and decryption; thus, it is slower than symmetric-key cryptography, and it is recommended to encipher small messages. However, to encrypt large messages, symmetric-key cryptography is preferable [2].

Symmetric-Key Encipherment

The general idea behind symmetric-key encipherment is that the same key is used for both the encryption and decryption processes. In the sender side, a plaintext is enciphered with a key (along with a set of functions and rules) and be sent as a ciphertext to the receiver. Upon receiving the ciphertext, the receiver deciphers it using the same key used for encrypting. Figure 2.1 shows the general architecture of symmetric-key cipher.

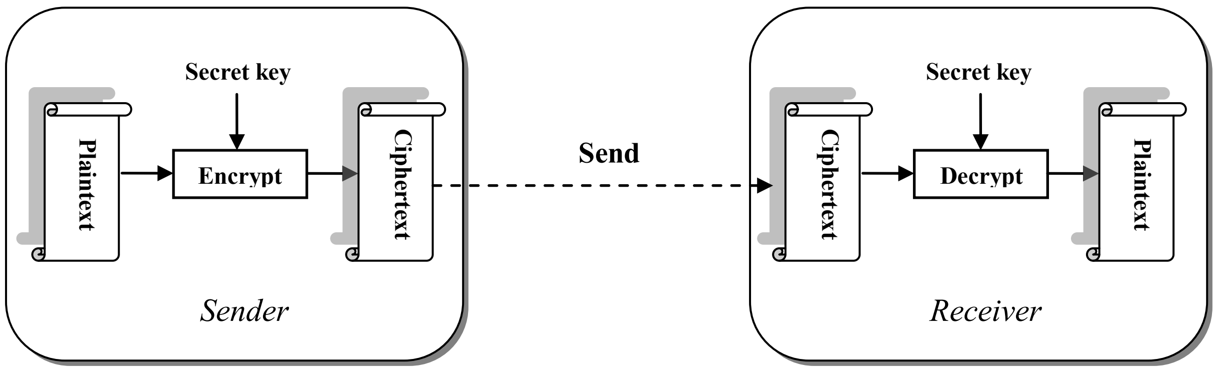


Figure 2.1 The schematic diagram of symmetric-key cipher.

Symmetric-key ciphers can be divided into two categories: traditional ciphers and modern ciphers [2]. There are two types of traditional symmetric-key ciphers: substitution ciphers, where each symbol in the plaintext is replaced by another symbol to generate the ciphertext, and transportation ciphers, where symbols in the plaintext exchange their values with each other to form the ciphertext. This type of cipher assumes that we always encrypt texts. With the advent of modern symmetric-key ciphers , other forms of data; such as videos, audios, images, can be fragmented into streams or blocks. Stream ciphers operate on a single bit at a time and the key and algorithm are applied to each bit in the data stream. Data is converted to bits and this simplifies the process of sending and receiving both the plaintext and ciphertext. But this cryptosystem is not as commonly used in modern cryptography as block ciphers. A symmetric-key modern block cipher divides the plaintext into n-bit blocks in which the key and algorithm are applied to these blocks rather than individual bits in the stream; detailed specifications can be found in [23].

One of the most commonly used algorithms that fall under symmetric-key cryptography is Data Encryption Standard (DES) [18], which was designed by IBM in responding of a request from the National Institute for Standards and Technology (NIST) to invent a national symmetric-key cryptosystem that was published in the 1970s [2]. DES is a symmetric-key block cipher that uses a 56-bit key length and a 64-bit block size. It is composed of complex set of encryption functions and transformations.

DES encryption process relies on the idea of permutations and rounds. Each 64-bit of the plaintext is permuted twice -initial and final permutations- according to a predefined rule. The permutations are used to change the order of plaintext symbols. DES operates 16 rounds for each block. Each round uses a different 48-bit key each of which is generated from a 56-bit cipher key. When the 64-bit input is initially permuted, it is processed separately in each round using the correspondent round key. After the 16^{th} round, the 64-bit output is subjected to the inverse initial permutation. DES decryption is the reverse approach of the encryption process.

Asymmetric-Key Encipherment

In asymmetric-key encipherment, a key pair of public key and private key is used. Mathematical calculations are done to bound the two keys in a certain relation. Any of the keys can be used for either encryption or decryption and the other is used for the reverse operation. For example, the public key is used to encrypt a message that can be decrypted only by the correspondent private key and vise-versa (see Figure 2.2). The advantage of this kind of cryptography is that even though the two keys are mathematically linked, it is not easy to calculate one of the keys with the knowledge of the other key.

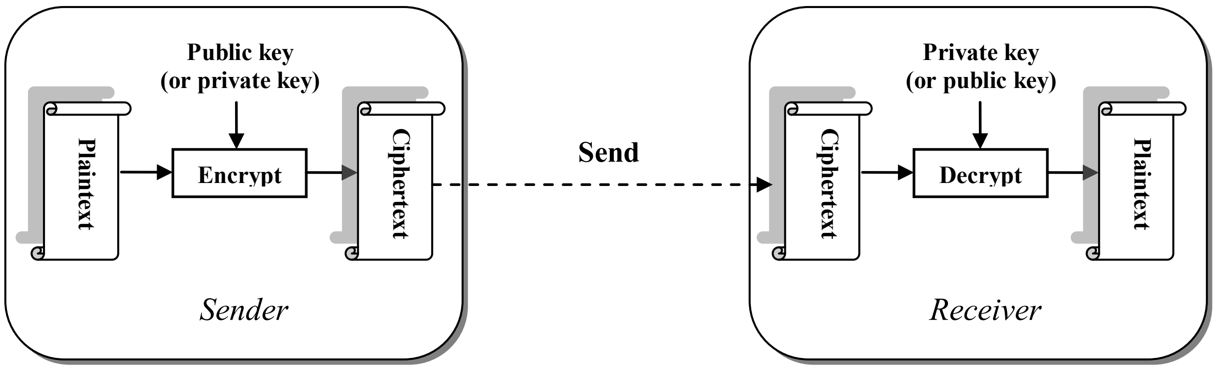


Figure 2.2 The schematic diagram of asymmetric-key cipher

In this cryptography, any involved party should have a key pair. The public key can be published to all parties whom the keys owner wants to share the key with. The private key is designed to be kept secretly for only the keys owner and not be distributed widely. If an outside party wants to send a message to the keys owner, she/he encrypts the message using the keys owner's public key. When receiving the ciphertext, the keys owner uses her/his private key to decrypt the message. In the opposite direction, if the keys owner wants to send an encrypted message to the outside party, she/he encrypts the message using her/his private key. The outside party decrypts the message using the owner's public key. The most known public-key cryptography algorithm is RSA [19][20], named after its inventors (Rivest, Shamir, and Adleman). Its security is based on the intractability of the integer factorization problem. RSA does not require complicated mathematical calculations, as is the case with DES. However, the strength of RSA cryptosystem comes from the difficulty of dealing with large numbers and in finding the prime factors of those large numbers.

Cryptographic Hash Functions

A cryptographic hash function is a one-way function that is used to convert a variable-length message to a fixed-length message which is called message digest [21]. Hashing functions support many applications and mainly used for authentication purposes. In addition, they can be used with databases to guarantee that information is not duplicate. Extensive studies have been dedicated to build unbreakable hash functions that are based on the compression concept. Message Digest (MD) and Secure Hash Algorithm (SHA) are two of the algorithms that prove their competitive performance. The standardized development of MD presented many versions including MD5. MD5 was designed by Ron Rivest in 1991 which is a compression function that divides a message into 512-bit blocks, padding is added as necessary to make the block length divisible by 512, and generates a 128-bit output.

Key Management

The principle problems with symmetric-key encipherment are key distribution and scalability. If Alice distributes her secret key among Bob, David and John and each one of them sends a decrypted message to Alice, there is no way that Alice could determine who is the sender; in other words, the message cannot be authenticated. To solve this problem, Alice needs a different key with each one of the three parties. But what if there are one thousand parties? Does Alice need to have one thousand keys? How would Alice maintain the distribution of this number of keys?

An efficient practical method to distribute keys is to use a third trusted party called Key Distribution Center (KDC) [2]. KDC allows each user to own one shared secret key with the KDC. If Bob wants to send a confidential message to Alice, he sends a request to KDC that he needs a shared secret key between him and Alice. The KDC asks Alice for permission. If Alice agrees, the KDC establishes the secret key and sends it to them.

In the previous sections, we discussed some cryptographic principles that are mainly used in the proposed protocol. The next section focuses on the subject of databases.

Databases

A database is an organized data structure used to store information. Databases are intended to operate large amounts of data. It allocates scattered information in one place so a user can easily access all of the information. The benefits of using databases includes shared access, minimal redundancy, data consistency, data integrity, and controlled access [3] [6]. A Database Management System (DBMS) is a software application to support all operations needed for databases. These operations include creating schemas, altering tables, modifying records, retrieving information and administrating a database. There are many DBMSs such as PostgreSQL, SQLite, Oracle and MySQL. MySQL was classified as one of the most widely used open-source Relational Database Management System (RDBMS) [22]. It provides a set of tools and commands to ease the interaction between users and data. It enables users to implement a database with tables, records and indexes. Moreover, it updates the indexes automatically as well as interprets SQL queries and establishes relations between tables. What makes MySQL competitive is its speed and ease of use. It is a powerful free open-source that is capable to handle large data sets.

The ultimate goal of the KCR protocol is using cryptography to secure large databases. All of the previously mentioned principles are used together to build the architecture of the proposed protocol. In the following chapter, I present in detail the KCR protocol.

CHAPTER 3

KCR Protocol

Efforts have been made to address the problem of compromised keys for large databases. However, a desirable protocol that has a powerful control over keys should be capable of simultaneously protecting the huge databases and minimizing the chance of re-encrypting such databases in case their keys have been attacked. In this chapter, I propose a protocol that achieves this objective.

General Description

In this study, I basically focus on key management for database encryption. The essential objective of KCR protocol is to solve the problem of re-encrypting a large database when its decryption key is compromised. Before I demonstrate the design details of KCR and how this protocol works, there are some assumptions and general explanations that should be taken into account:

- KCR protocol uses one key for encryption and a set of keys for decryption. When any of the decryption keys is compromised, it is discarded and replaced with another decryption key without the need of re-encrypting the large database.
- Clients are grouped. Each group is granted a decryption key that is going to be used by all clients in the group. When a key is compromised, any of the group members contacts the responsible agency then gets a new decryption key for her/his group. The protocol assumes that the groups are predefined and the number of groups is less than the number of decryption keys.
- Distribute control in the context of decryption keys among different parties is proposed as a solution to strengthen the security measures of large database

contents. The primary cryptographic algorithm of KCR protocol is RSA which demands two keys: public key and private key. The public key is used for database encryption while the private key is used for database decryption. The private key is divided into a set of key pairs in which the product of the two entries of each key pair equals the original private key; in other words, the entries represent factors of the original private key. These entries are distributed between the database server and the groups.

- To decrypt a database object, the workload of the decryption process is distributed between the client who wants to access the object and the database server.
- The protocol provides secure communications between involved parties.

KCR Encryption and Decryption

Essentially, KCR protocol is based on RSA cryptosystem which consists of three processes: keys generation, encryption, and decryption. The proposed protocol adds more features to these processes.

Keys Generation

The public and private keys are generated as follows. First, a server selects two large primes, P and Q. Then, it computes $N = PQ$ and the totient function of N which is $ø(N) = (P-1)(Q-1)$. The server selects randomly an integer number E such that the greatest common devisor of D and $ø(N)$ equals 1. An integer number D is calculated as follows: $D = E^{-1} \mod ø(N)$. Now, the keys are ready to be used, the public key is (E, N) and the private key is (D, N). The server computes the factors of D: $F_1, F_2, F_3, \ldots, F_n$. Then, it generates a set of key pairs $\{(D_1, D_{1'}), (D_2, D_{2'}), (D_3, D_{3'}), \ldots, (D_n, D_{n'})\}$ in which $D = D_1D_{1'} = D_2D_{2'} = D_3D_{3'} = \ldots = D_nD_{n'}$.

Encryption

A plaintext P is encrypted to generate a ciphertext C using the public key (E, N) as follows: $C = P^E \bmod N$.

Decryption

To get the original plaintext P, partial decryption is done on the ciphertext C using the key pair $(D_i, D_{i'})$ and the modulus N: $C' = C^{D_i} \bmod N$, $P = (C')^{D_{i'}} \bmod N$. In case the key pair $(D_i, D_{i'})$ is compromised, it must be discarded and replaced by a new key pair $(D_j, D_{j'})$. To get P using the new key pair, the partial decryption process is as follows: $C' = C^{D_{j'}} \bmod N$, $P = (C')^{D_j} \bmod N$.

Proof of KCR

The following proves that encryption and decryption are inverses of each other:

We want to prove that: $P = C^D \bmod N$ where $D = D_i D_{i'}$

$P = (C')^{D_{i'}} \bmod N$

$= (C^{D_i})^{D_{i'}} \bmod N$ // $C' = C^{D_i} \bmod N$

$= (C)^{D_i D_{i'}} \bmod N$ // $((X)^y)^z = (X)^{yz}$

$= C^D \bmod N$ // $D = D_i D_{i'}$

A summary of all notations used in KCR encryption and decryption procedures is shown in Table I.

TABLE I

NOTATIONS USED IN KCR ENCRYPTION AND DECRYPTION

P, Q	Two large primes
N	Modulus
(E, N)	Public key
(D, N)	Private key
F_i	The i^{th} private key factor
(D_i, $D_{i'}$)	The i^{th} private key pair
D_i	The first entry of the i^{th} key pair
$D_{i'}$	The second entry of the i^{th} key pair
P	Plaintext
C	Fully decrypted ciphertext
C'	Partially decrypted ciphertext

Design

The design architecture of KCR protocol comprises four components: Keys server, KDC server, Database server, and Client.

Components

The basic idea that explains the tasks of each component is as follows:

1) Keys server: Keys server is responsible of KCR keys generation process. It calculates the RSA public key *(E, N)* and private key *(D, N)* . When this process is accomplished, the server sends the public key *(E, N)* to the Database server. In addition, it stores the private key pairs $\{(D_1, D_{1'}), (D_2, D_{2'}), (D_3, D_{3'}), \ldots, (D_n, D_{n'})\}$ in its database, and it maintains all information regarding these pairs. An important information that Keys server should store in its database is key pairs status K_{status}. The initial value of key pairs status is "inactive". When a key is granted, the status value is changed to "active". When a key is compromised, the status is replaced with the value "compromised". In addition to these tasks, Keys server is primarily responsible of the key distribution process. When a key pair first entry D_i is granted to a group G_i , Keys server forwards the key pair second entry $D_{i'}$ to the Database server along with information about the group which owns the key.
2) KDC server: KDC forms groups, which are predefined, for clients then it distributes those clients (as members) among the different groups according to predefined rules. The server contains a database to store information of the groups $(G_1, G_2, G_3, \ldots, G_n)$. The following example illustrates this approach. In a dentists clinic, suppose there are *d* dentists, *a* dentist assistants, and *p* patients. So, the groups are dentist, dentist assistant and patient, and the total number of clients is $d+a+p$ in

which each one of the clients is a member of her/his correspondent group. Other than forming groups, KDC is needed to manage the interaction between groups members and the Keys server. It acts as a trusted agency to send confidential messages from members to Keys server and vise versa. Also, this server is used to associate each group with a key. It establishes a connection with the Keys server to obtain the first entry of key pairs $(D_1, D_2, D_3, \ldots, D_n)$ and grants each group a key.

3) Client: When a client wants to process a query to access a database object, she/he needs to join a group in the first place. She/he contacts the KDC server to join a group G_i. Then, the KDC sends a member ID M_i_ID and the key D_i to the client. I assume that the group IDs are published. To process a query Q_i , the client sends the ID of group G_i and Q_i to the Database server. After receiving the partially decrypted object from the Database server, the client finishes the decryption process by using her/his group's key D_i. Moreover, the client is responsible to inform KDC when the key is compromised. Consequently, KDC contacts Keys server, then the latter server updates the information stored in its database regarding the compromised key. After that, Keys server gives a new decryption key ,if available, to the group. Accordingly, Keys server forwards these updates to the Database server.
4) Database server: This server contains the main database that we need to encrypt. The encipherment is done using the RSA public key received from Keys server. In addition to this task, the server plays an important role in the decryption process. When a client sends a query Q_i to Database server along with other information to identify the group she/he belongs to, the server uses the appropriate key received from Keys server $D_{i'}$ and partially decrypts the object and sends the result to the

client. Then, the client completes the decryption process by decrypting the received result using her/his key D_i.

Scenarios

As aforementioned, KCR protocol is based on the interaction between four essential components: Keys server, KDC server, Database server and Client. These interactions define five important scenarios: Public key distribution scenario, private key distribution in normal case scenario, join group scenario, private key distribution in case a key is compromised scenario, access database scenario. The following presents details of each scenario (Hint: M, M', $E(\,)$, $D(\,)$, and $Hash(\,)$ denotes to transmitted message, encrypted message, encryption function, decryption function, and hash function, respectively).

Scenario 1 - Public key distribution. After generating the key pairs (public and private key), Keys server sends a message that includes the public key to Database server. The message, $M = (E, N)$, is encrypted using a secret session key K_L between Keys server and Database server: $M' = E(M, K_L)$. Upon receiving the encrypted message, Database server decrypts it with K_L: $M = D(M', K_L)$ to get the public key and encrypts the database according to the KCR encryption process discussed earlier. The design architecture of this scenario is shown in Figure 3.1. Table II shows the notations used in the scenario.

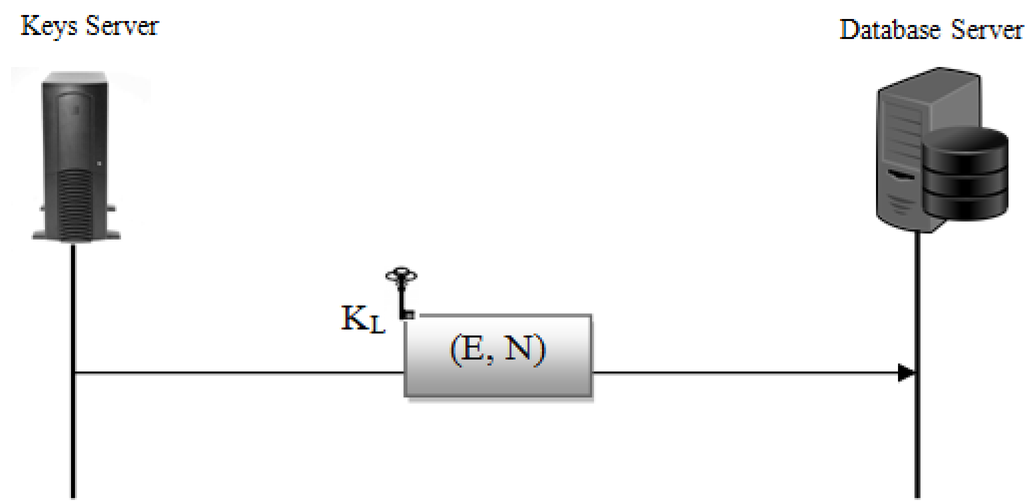


Figure 3.1. KCR protocol: Scenario 1 - Public key distribution

TABLE II

NOTATIONS USED IN SCENARIO 1

K_L	Encrypted with Keys-Database secret key
(E, N)	Public key for database encryption

Scenario 2 - Private key distribution in normal case. This scenario consists of three steps:

1) To grant a decryption key to a group G_i, the KDC creates a message that contains the group ID G_i_ID and the key status, $M = (G_i\_ID, K_{status})$. The key status is "inactive" to indicate that group G_i is a new group and does not own a key yet. The message is sent to Keys server. The message is encrypted using a secret session key K_M between KDC server and Keys server: $M' = E(M, K_M)$.

2) When Keys server receives the encrypted message *M'*, it decrypts it: $M = D(M', K_M)$ to get G_i_ID and K_{status}. Since K_{status} = "inactive", Keys server scans its database to extract a key pair $(D_i, D_{i'})$ then grants this key pair to the group G_i . Keys server updates the value of the key pair status which is stored in its database from "inactive" to "active". The server then replies to KDC by sending a message that contains the first entry of the private key pair (*N* is part of the private key); $M =(D_i, N)$. For confidentiality, the message is encrypted using the secret session key K_M as follows: $M' = E(M, K_M)$.

3) Keys server forwards a message to Database server. The message includes the second entry of the private key pair, G_i_ID, and the hash value of the first entry of the private key; $M = ((D_{i'}, N), G_i\_ID, Hash(D_i))$. The message is encrypted with a secret session key K_L shared between Keys server and Database server. The format of the message is as follows: $M' = E(M, K_L)$.

Figure 3.2 shows the scenario steps. Table III summarizes the notations used in the scenario.

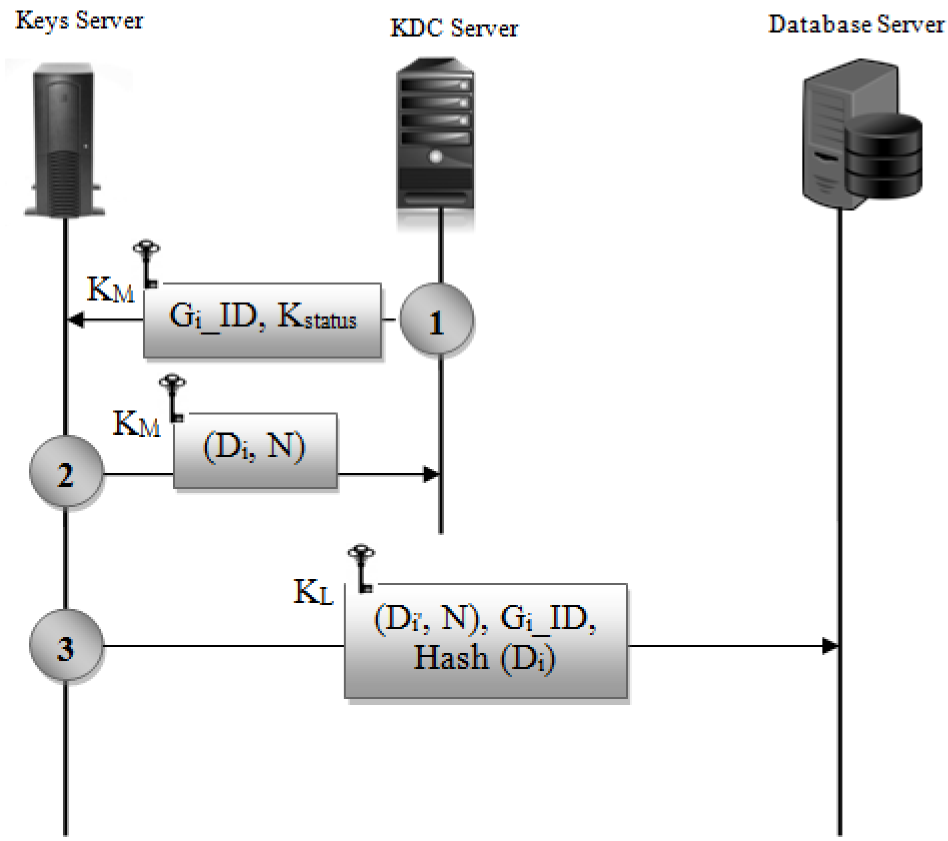


Figure 3.2. KCR protocol: Scenario 2 - Private key distribution in normal case

TABLE III

NOTATIONS USED IN SCENARIO 2

 K_L	Encrypted with Keys-Database secret key
 K_M	Encrypted with KDC-Keys secret key
G_i	The i^{th} group
G_i_ID	The ID of group G_i
K_{status}	Key status (active, inactive, compromised)
(D_i, N)	The i^{th} first entry of the i^{th} private key pair
$(D_{i'}, N)$	The i^{th} second entry of the i^{th} private key pair
Hash(D_i)	Hash value of D_i

Scenario 3 - Join group. The following steps explain how a client joins a group:

1) A client sends a short message that is encrypted by a secret session key K_N between her/him and the KDC server. The message contains the group ID G_i_ID of the group G_i that she/he wants to join, and it is encrypted as follows: $M' = E(G_i\_ID, K_N)$.

2) KDC server decrypts the message as follows: $M = D(M', K_N)$ to get G_i_ID. Then, it adds the client as a member M_i to group G_i and responds by sending the member ID M_i_ID and the group's key; $M =(M_i\_ID,(D_i, N))$. The message is sent as an

encrypted message: $M' = E(M, K_N)$.Then the client decrypts the message: $M = D(M', K_N)$.

The steps are shown in Figure 3.3. The notations used in the scenario are summarized in Table IV.

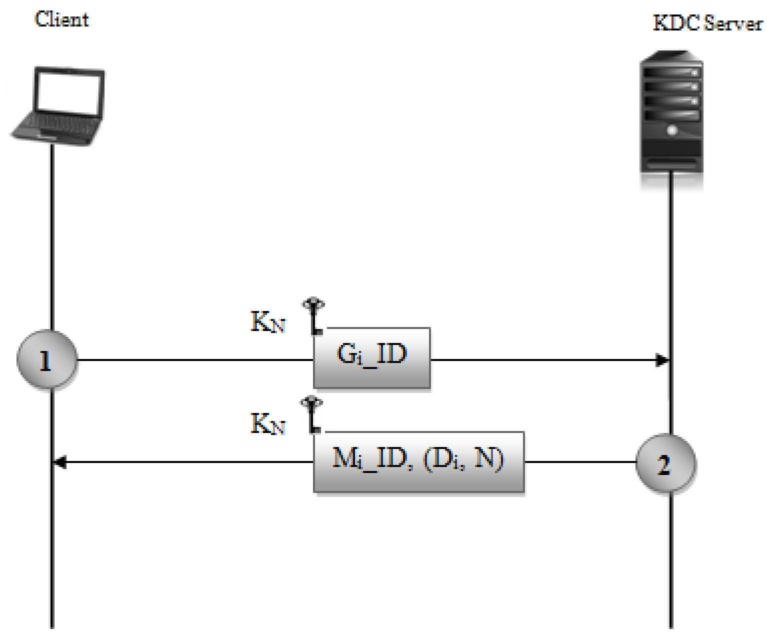


Figure 3.3. KCR protocol: Scenario 3 - Join group

TABLE IV

NOTATIONS USED IN SCENARIO 3

 K_N	Encrypted with Client-KDC secret key
G_i	The i^{th} group
G_i_ID	The ID of group G_i
(D_i, N)	The i^{th} first entry of the i^{th} private key pair
M_i	The i^{th} member of group G_i
M_i_ID	The ID of member M_i
(D_i, N)	The i^{th} first entry of the i^{th} private key pair

Scenario 4 - Private key distribution in case a key is compromised. The heart of the KCR protocol is handling the situation when a large database decryption key is compromised. This scenario presents the steps that explain the interaction between the involved components:

1) When a private key pair $(D_i, D_{i'})$ of a group G_i has been attacked, group members are responsible of informing the KDC about the situation. A client, a group member, sends a message to the KDC server that contains G_i_ID and K_{status} = "compromised" : $M = (G_i\_ID, K_{status})$. The message is encrypted using a secret session key K_N between the client and KDC server: $M' = E(M, K_N)$.
2) When KDC receives the encrypted message M, it decrypts it: $M = D(M', K_N)$ to get G_i_ID and K_{status}. Since K_{status} = "compromised", KDC forwards the message to

Keys server to get a new key for group G_i. The message is encrypted with K_M, a secret key shared between KDC and Keys servers: $M' = E(M, K_M)$.

3) Then, Keys server decrypts message M': $M = D(M', K_M)$ and updates its database that contains information about group G_i and its key pair $(D_i, D_{i'}$; it changes the value of the key pair status from "active" to "compromised". The server then grants another key pair $(D_j, D_{j'})$ to group G_i. It sends a message containing the first entry D_j of the new key pair as a private key $(M = (D_j, N))$ to KDC after encrypting it using K_M : $M' = E(M, K_M)$.

4) After receiving the message from Keys server, KDC decrypts it: $M = D(M', K_M)$ to get the key (D_j, N) then grants the new decryption key to group G_i. Next, the server sends an enciphered message using K_N to the client. The message includes the new key D_j: $M' = E(M, K_N)$.

5) Keys server sends the second entry $D_{j'}$ to Database server. The message $(M = (D_{j'}, N), G_i\_ID , Hash(D_j)))$ is encrypted by K_L: $M' = E(M, K_L)$.Consequently, Database server discards the old key of group G_i and exchanges it with the new one.

The steps of this scenario are shown in Figure 3.4. the notations used are summarized in Table V.

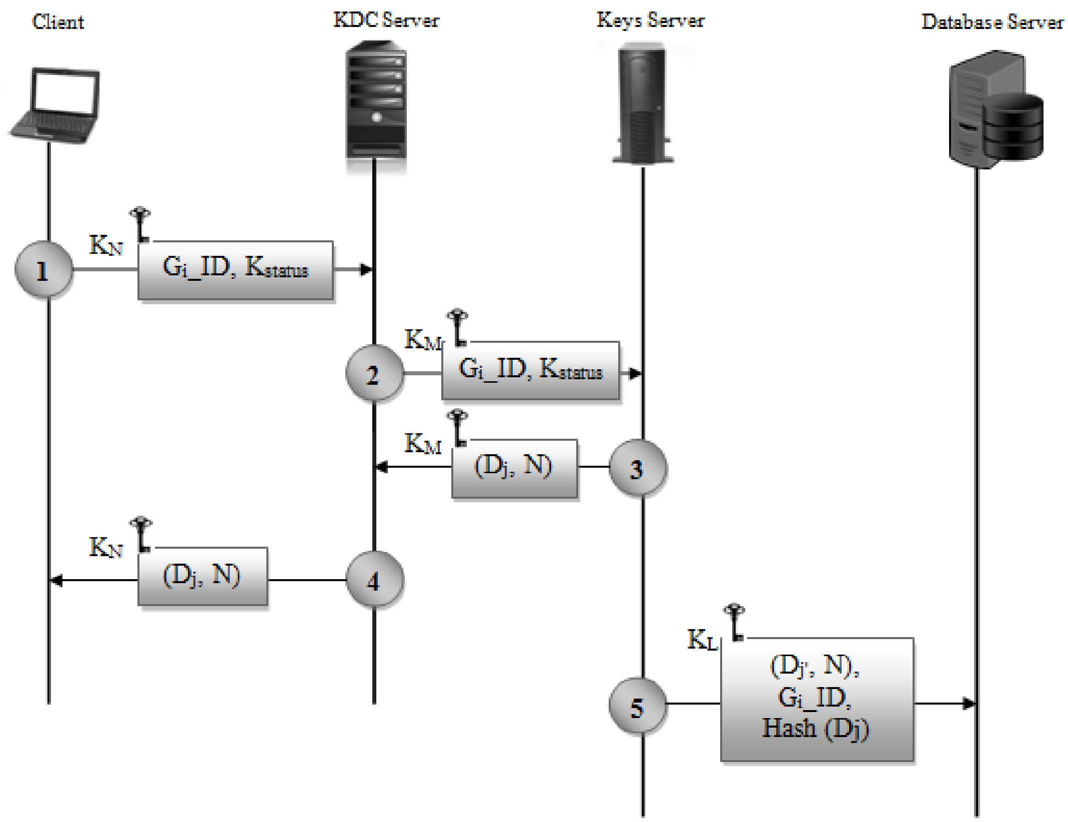


Figure 3.4. KCR protocol: Scenario 4 - Private key distribution in case a key is compromised

TABLE V

NOTATIONS USED IN SCENARIO 4

 K_L	Encrypted with Keys-Database secret key
 K_M	Encrypted with KDC-Keys secret key
 K_N	Encrypted with Client-KDC secret key
G_i	The i^{th} group
G_i_ID	The ID of group G_i
K_{status}	Key status (active, inactive, compromised)
(D_j, N)	The j^{th} first entry of the j^{th} private key pair
$(D_{j'}, N)$	The j^{th} second entry of the j^{th} private key pair
$Hash(D_j)$	Hash value of D_j

Scenario 5 - Access database. The scenario steps are explained as follows:

1) To process a query Q_i, a client contacts the Database server and sends a message that contains Q_i along with an additional information that helps the Database server indicate which key to use. This information is either the group ID *Gi_ID* or the hash value of the group's key first entry D_i. If the client chooses G_i_ID, the message content is $M = (Q_i, G_i\_ID)$. If she/he forgets her/his group ID, she/he can still access the database using the hash value of her/his key. In this case, the message

content is $M = (Q_i, Hash(D_{i}))$. A secret session key K_O is used to encrypt the transmitted message: $M' = E(M, K_O)$.

2) Upon receiving the query, Database server decrypts the message: $M = D(M', K_O)$, processes the query Q_i, and extracts the second entry $D_{i'}$ of the private key pair. Then, it decrypts the encrypted query result CR_i to generate a partially decrypted ciphertext: $C' = (CR_i)^{D_{i'}} \bmod N$. Database server sends an encrypted message to the client using K_O. The message includes the partially decrypted result C': $M' = E(C', K_O)$. When the client receives the message, she/he decrypts it: $M = D(M', K_O)$ to get C' then completes the decryption process to get the original Result R_i:
$R_i = (C')^{D_i} \bmod N$.

Figure 3.5 shows the steps of database access scenario with G_i_ID choice. Figure 3.6 shows the steps of the scenario with hash value choice. Table VI summarizes the notations used in this scenario.

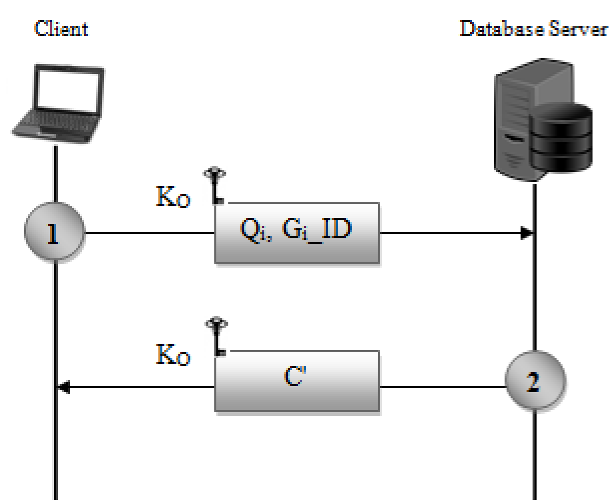


Figure 3.5. KCR protocol: Scenario 5 - Database access with group ID choice

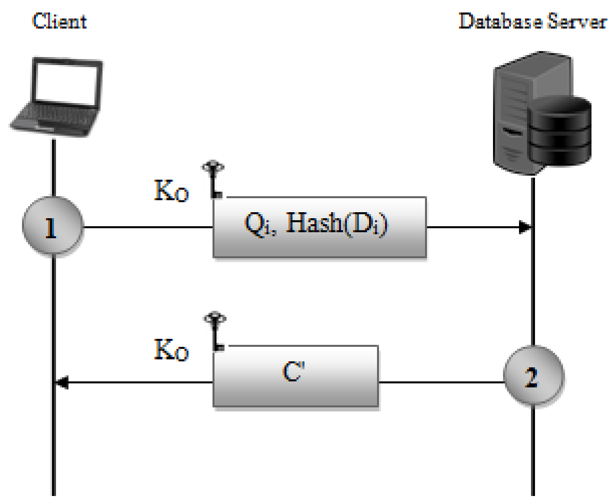


Figure 3.6. KCR protocol: Scenario 5 - Database access with hash value choice

TABLE VI

NOTATIONS USED IN SCENARIO 5

K_O	Encrypted with client-Database secret key
G_i	The i^{th} group
Q_i	The i^{th} client requested query
(D_i, N)	The i^{th} first entry of the i^{th} private key pair
$(D_{i'}, N)$	The i^{th} second entry of the i^{th} private key pair
$Hash(D_i)$	Hash value of D_i
CR_i	The i^{th} encrypted result of query Q_i
R_i	The i^{th} result of query Q_i after decryption
C'	The partially decrypted cipher text

Further Discussion

Two issues are worth more discussion. The first is that, in practice, the decryption key pairs that are derived from the original private key are limited in number. This limitation may cause the protocol not to accommodate a high volume of clients. The following example explains this problem. Suppose there are X decryption key pairs and Y clients where $X < Y$. By granting each client a key pair, the number of clients who own a decryption key pair would be X while $Y-X$ would remain with no key pairs and would not be able to access the database objects. This approach is not practical. The proposed solution assumes that the number of key pairs is greater than the number of clients. Besides this assumption, clients are distributed into groups in which each group is granted a different key pair. When a client needs to access a database object, she/he uses the key of the group that she/he is a member of. In the event that a key is compromised, all servers in KCR protocol must be informed by any member of the group who owns the key about this case. Accordingly, the component that is responsible of private key pairs generation and distribution, Keys server, gives a new key pair to the group, and the old key pair would be discarded.

The second is that, in KCR protocol, each group must have a decryption key pair. When a key pair of a group is compromised and the Keys server is out of key pairs, the group will not be granted a new key pair. Since this situation is not acceptable in real life, the databases needs to be re-encrypted. Therefore, new public and private keys must be generated. In addition, all existing key pairs that are derived from the old private key should be discarded and be replaced by a new set of key pairs that are derived from the new private key.

To summarize this chapter, I discussed in detail the architecture of KCR protocol. I also defined the basic operations which are keys generation, encryption and decryption. I further presented the five scenarios and their design architecture. Some issues have been discussed to add more details to KCR protocol. The protocol is implemented to test its efficiency in a real environment. The implementation details are discussed in next chapter.

CHAPTER 4

Implementation

In the previous chapter, I have defined the design of KCR Protocol. I also explained in detail how the components are communicated with each other and the format of transmitted messages. This chapter presents detailed implementation of the proposed protocol. It explores other issues related to the implementation stage. The implementation is not complicated; it simply demonstrates the basic idea of KCR protocol. Although the protocol is proposed to solve the problem of compromised key in large databases, the size of the database used is very small.

Platform

The hardware used to implement the project has the following characteristics:

1) Laptop: Sony VAIO VPCEA.
2) Processor: Intel(R) Core(TM) i5 CPU, M 460, 2.53GHz.
3) Memory (RAM): 6.00 GB.
4) System type: 64-bit Operating System.
5) Operating System: Windows 7 Home Premium.

The project uses a set of third-party libraries a long with software programs and applications to support the runtime environment:

1) Programming language: Java.
2) Java Runtime Environment version 7.0.
3) DBMS: MySQL
4) MySQL client software: Workbench 6.0.

5) Integrated Device Electronics (IDE): Eclipse Java EE IDE for Web Developers version 2.0.1.20130919-0803.
6) Driver for MySQL: Java Database Connectivity (JDBC) that is a technology to address the interaction between users and databases. It is used as an API for the Java programming language. See Figure 4.1.

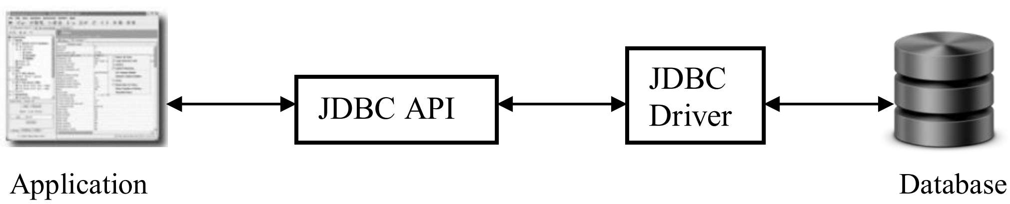


Figure 4.1 JDBC architecture

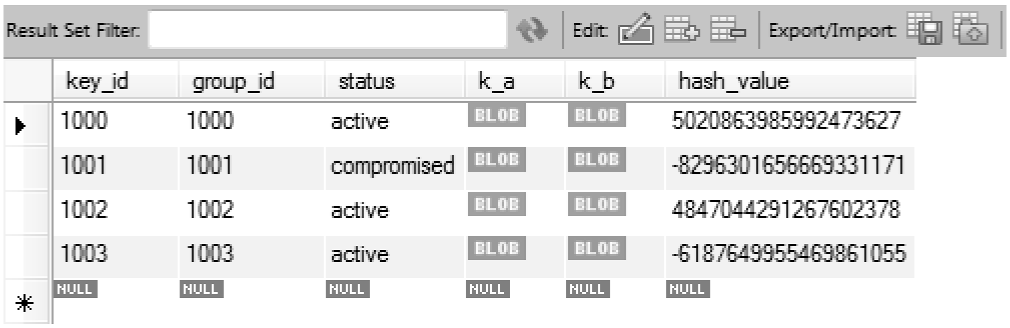
Implementation Details

The implementation analyzes the details described in the design architecture of KCR protocol. It presents the database schemas used in the protocol and demonstrates how the scenarios are integrated together.

Database Schemas

As aforementioned, there are four components involved in KCR protocol: Keys server, KDC server, Database server and Client. Each one of the first three components has a database to store certain information. The following describes the database schemas of each component:

- Keys server database schema: The database schema, named "keysserver", of Keys server stores information regarding the private key pairs. It contains only one table named "keypairs". Each record in the table includes information about a single private key pair. The record consists of six columns: key pair ID (key_id), key pair group id(group_id), key pair status(status), key pair first entry(k_a), key pair second entry(k_b), and key pair first entry hash value(hash_value). When a key pair is generated, the first entry is hashed using MD5 hash function. Figure 4.2 shows an example of "keypairs" table.



key_id	group_id	status	k_a	k_b	hash_value
1000	1000	active	BLOB	BLOB	5020863985992473627
1001	1001	compromised	BLOB	BLOB	-8296301656669331171
1002	1002	active	BLOB	BLOB	4847044291267602378
1003	1003	active	BLOB	BLOB	-6187649955469861055
NULL	NULL	NULL	NULL	NULL	NULL

Figure 4.2 An example of "keypairs" table

- KDC server database schema: The database schema is "kdc" which has two tables. The first is called "groups" that holds information of five groups. Each group has an ID, name, key and key status. The two latter represent information relating to the first entry of the key pair that the group owns. Figure 4.3 shows an example of this table. The second table in the database schema is named "members" which represents information about members of groups. The columns are member ID, group ID, member name, and password. Figure 4.4 shows an example of "members" table.

group_id	group_name	key_status	k_a
1000	group00001	active	BLOB
1001	group00002	compromised	BLOB
1002	group00003	active	BLOB
1003	group00004	active	BLOB
1004	group00005	compromised	BLOB
NULL	NULL	NULL	NULL

Figure 4.3 An example of "groups" table

Result Set Filter: Edit: Export/Import: Wrap Cell Content:

member_id	group_id	member_name	member_password
1000	1001	Fatemah	abcd
1001	1000	Amal	abcd
1002	1004	Abrar	abcd
NULL	NULL	NULL	NULL

Figure 4.4 An example of "members" table

- Database server database schema: In the implementation, I use a small database that contains information about a zoo. The schema is named "zoo", and it has three tables: "animals", "animals_", and "keyinfo". Data about animals are stored in "animals" table. It divides animals according to their category. Each category has an ID, a name, and a total number of animals. Figure 4.5 shows an example of "animals" table. The basic unit of information in a database is the record. Thus, the encryption procedure of "animals" table should be record oriented. To effectively encrypt a record, all of its fields should be encrypted individually forming a sequence of encrypted fields. The encrypted records are stored in the table "animals_" with the same columns names of "animals" table. An example of "animals_" table is shown in Figure 4.6. The last table is named "keyinfo". It is used to store information regarding the second entries of decryption key pairs that are granted to groups. The columns are key ID, group ID, key status, key pair second entry, hash value of key pair first entry. Figure 4.7 shows an example of this table.

id	category	total
1000	Lion	10
1001	tiger	20
1002	Horse	30
1003	Bear	40
NULL	NULL	NULL

Figure 4.5 An example of "animals" table

id	category	total
1000	BLOB	BLOB
1001	BLOB	BLOB
1002	BLOB	BLOB
1003	BLOB	BLOB
NULL	NULL	NULL

Figure 4.6 An example of "animals_" table

key_id	group_id	status	k_b	hash_value
1000	1000	active	BLOB	5020863985992473627
1002	1002	active	BLOB	4847044291267602378
1003	1003	active	BLOB	-6187649955469861055
1004	1004	active	BLOB	0
1005	1001	active	BLOB	0
NULL	NULL	NULL	NULL	NULL

Figure 4.7 An example of "keyinfo" table

Keys Generation

The first task of Keys server is keys generation. When public and private keys are generated using RSA algorithm, they are stored in separate files with file extension ".rsa". They can be kept secretly in either external hard disk or Keys server local machine. Next, private key pairs need to be derived from the original private key. First we need to find factors of the original private key. To accomplish this process, I use an open-source code [24] that can find up to 30-digit factors of numbers up to 1000 digit long. After computing the factors, they are stored in a file with file extension ".rsa". The next step is calculating the private key pairs. As aforementioned, the product of the two entries of a key pair should be equal to the original private key. The pseudocode of private key pairs generation is as follows:

1) Select an array A to store private key pairs.
2) For a key pair (D_i, $D_{i'}$), select K_1 and K_2 to represent the first entry and second entry, respectively.
3) For each factor F_i ($i = 1,2,3, \ldots, n$ where n is the number of factors), do the following:
   a. Let $K_1 = F_i$ and $K_2 = 1$.
   b. For each factor F_j ($j = 1,2,3,\ldots,n$ where $i \neq j$), multiply F_j by K_2. The multiplication process is done $n-1$ times.
   c. Select a variable number M where $M = max(K_1, K_2)$ (where max is a function that returns the highest value of two numbers).
   d. Select another variable number T such that $T = D\%M$ (where D is the original private key).
   e. If $T = 0$, generate a key pair $KP = (K_1, K_2)$.
   f. Scan array A to check if it contains a key pair with same value as KP. If it contains the same key pair, skip and go to step (g); otherwise, add KP to the array.

g. For each factor F_k ($k = 1,2,3, \ldots, n$ where n is the number of factors), do the following:

   i. Multiply F_k by K_1, and let $K_2 = 1$.
   ii. For each factor F_u (where $u = k+1$) multiply F_u by K_2.
   iii. Select a variable number $P = K_1K_2$.
   iv. Select a variable number W such that W = D%P.
   v. If W = 0, generate a key pair $KR = (K_1, K_2)$.
   vi. Scan array A to check if it contains a key pair with same value as *KR*. If it contains the same key pair, skip; otherwise, add KR to the array.

4) Scan array A to check that for each key pair $(D_i, D_{i'})$ there exist a key pair $(D_{i'}, D_i)$; otherwise, generate a key pair $(D_{i'}, D_i)$ and add it to the array.

Now that all possible private key pairs are generated, we need to store them in the Keys server database; particularly in "keypairs" table. All key pairs statuses should be set to "inactive" since no key is granted to a group yet. Figure 4.8 shows an example of "keypairs" table with 38 key pairs.

Result Set Filter: Edit: Export/Import:

key_id	group_id	status	k_a	k_b	hash_value
1	NULL	inactive	BLOB	BLOB	-6831124328378434052
2	NULL	inactive	BLOB	BLOB	2155982158824764557
3	NULL	inactive	BLOB	BLOB	-8722224433160790119
4	NULL	inactive	BLOB	BLOB	-6494578374345099477
5	NULL	inactive	BLOB	BLOB	7716181922073026682
6	NULL	inactive	BLOB	BLOB	-6803601381396060957
7	NULL	inactive	BLOB	BLOB	7640699948786602131
8	NULL	inactive	BLOB	BLOB	-5784801114835674760
9	NULL	inactive	BLOB	BLOB	-9033490034240231277
10	NULL	inactive	BLOB	BLOB	-7012721511857543887
11	NULL	inactive	BLOB	BLOB	-2735182576426872571
12	NULL	inactive	BLOB	BLOB	1444063446025208723
13	NULL	inactive	BLOB	BLOB	-1761960517980859454
14	NULL	inactive	BLOB	BLOB	3696563988952405018
15	NULL	inactive	BLOB	BLOB	241600442490808180
16	NULL	inactive	BLOB	BLOB	660048615099099698
17	NULL	inactive	BLOB	BLOB	-329665467061696787
18	NULL	inactive	BLOB	BLOB	8219911656708429919
19	NULL	inactive	BLOB	BLOB	8623951624465618688
20	NULL	inactive	BLOB	BLOB	-8425020461600410724
21	NULL	inactive	BLOB	BLOB	1982018180544787479
22	NULL	inactive	BLOB	BLOB	-6187649955469861055
23	NULL	inactive	BLOB	BLOB	-1904683017053474758
24	NULL	inactive	BLOB	BLOB	-5189928506646213691
25	NULL	inactive	BLOB	BLOB	-6329672147696771449
26	NULL	inactive	BLOB	BLOB	-1152668957698890062
27	NULL	inactive	BLOB	BLOB	-8066103964492370758
28	NULL	inactive	BLOB	BLOB	-6859501549342179320
29	NULL	inactive	BLOB	BLOB	2095263708004682852
30	NULL	inactive	BLOB	BLOB	-3609778160197642217
31	NULL	inactive	BLOB	BLOB	-1938169658791457844
32	NULL	inactive	BLOB	BLOB	-6257339918928147811
33	NULL	inactive	BLOB	BLOB	-699034747928976105
34	NULL	inactive	BLOB	BLOB	-8994101046912019253
35	NULL	inactive	BLOB	BLOB	-3395633363429044118
36	NULL	inactive	BLOB	BLOB	3176838218257335284
37	NULL	inactive	BLOB	BLOB	-8751729337321694785
38	NULL	inactive	BLOB	BLOB	-9145444701133060395
NULL	NULL	NULL	NULL	NULL	NULL

Figure 4.8 "keypairs" table before any key is granted

After we have seen how the keys generation process is processed, let us now see how the scenarios discussed earlier in chapter 3 are implemented.

Scenario 1 - Public Key Distribution

The implementation of this scenario is straightforward. Once a public key is generated, it is sent to the Database server and stored in a file with file extension ".rsa". To protect the key from any attacks, this file is either kept safe in an external hard disk or stored in the Database server local machine. The public key is used to encrypt the table "animals" in the database schema "zoo". The encrypted records are stored in table "animals_".

Scenario 2 - Private Key Distribution in Normal Case

KDC server creates groups and stores their information in its database. Then, it contacts Keys server to grant a key pair for each group. Since there are five predefined groups, the first five key pairs are granted sequentially (See Figure 4.9, Figure 4.10, and Figure 4.11).

Result Set Filter: Edit: Export/Import: Wrap Cell Content:

key_id	group_id	status	k_a	k_b	hash_value
1	1000	active	BLOB	BLOB	-6831124328378434052
2	1001	active	BLOB	BLOB	2155982158824764557
3	1002	active	BLOB	BLOB	-8722224433160790119
4	1003	active	BLOB	BLOB	-6494578374345099477
5	1004	active	BLOB	BLOB	7716181922073026682
6	NULL	inactive	BLOB	BLOB	-6803601381396060957
7	NULL	inactive	BLOB	BLOB	7640699948786602131
8	NULL	inactive	BLOB	BLOB	-5784801114835674760
9	NULL	inactive	BLOB	BLOB	-9033490034240231277
10	NULL	inactive	BLOB	BLOB	-7012721511857543887
11	NULL	inactive	BLOB	BLOB	-2735182576426872571
12	NULL	inactive	BLOB	BLOB	1444063446025208723

Figure 4.9 Table "keypairs" after granting five key pairs to the five groups

group_id	group_name	key_status	k_a
1000	group00001	active	BLOB
1001	group00002	active	BLOB
1002	group00003	active	BLOB
1003	group00004	active	BLOB
1004	group00005	active	BLOB
NULL	NULL	NULL	NULL

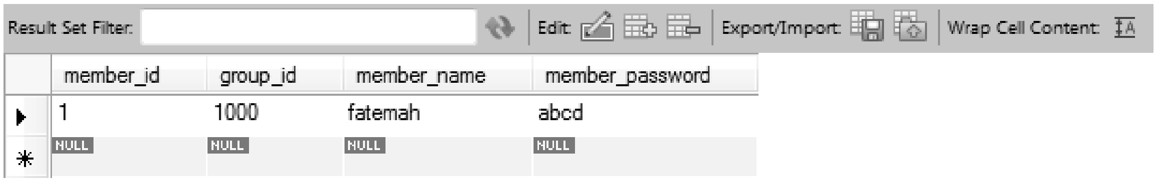
Figure 4.10 Table "groups" after granting key pairs to the five group

key_id	group_id	status	k_b	hash_value
1000	1000	active	BLOB	-6831124328378434052
1001	1001	active	BLOB	2155982158824764557
1002	1002	active	BLOB	-8722224433160790119
1003	1003	active	BLOB	-6494578374345099477
1004	1004	active	BLOB	7716181922073026682
NULL	NULL	NULL	NULL	NULL

Figure 4.11 Table "keyinfo" after granting key pairs to the five groups

Scenario 3 - Join Group

When a client wants to join a group, she/he needs to sign up for KDC server. She/he sends the group ID, a username, and a password. Then, KDC creates an account for the client and adds her/him as a member to the specified group (See Figure 4.12). KDC returns the client's member ID to the client. When a client needs to get a decryption key, she/he sends her/his member ID, group ID, username, password, and "active" as the key's status to KDC server. After that, KDC verifies client's information. If she/he is an authorized client, KDC replies to her/him by sending the group's key. After receiving the key, the client stores it in a file (the file extension is ".rsa").



Result Set Filter: Edit: Export/Import: Wrap Cell Content:

member_id	group_id	member_name	member_password
1	1000	fatemah	abcd
NULL	NULL	NULL	NULL

Figure 4.12 Table "members" after creating an account for a client. The client joined group "group00001"

Scenario 4 - Private Key Distribution in Case a Key is Compromised

When a group's key is compromised, any of its members notifies KDC about this matter. A member (a client) sends her/his member ID, group ID, username, password, and "compromised" as the key's status to KDC server. After that, KDC verifies client's information. If she/he is an authorized client, KDC notifies Keys server about this issue. Consequently, Keys server updates its database and gives a new key pair to the specified group (See Figure 4.13), sends the first entry of the key pair to KDC, and forwards a message to Database server stating that the key of that group has been compromised and a new key pair has been granted. Database server discards the old key and replaces it with the new key received from Keys server. After receiving the key from Keys server, KDC updates its database then replies to the member by sending the group's key. After receiving the key, the client stores it in a file (the file extension is ".rsa").

Result Set Filter: | Edit: | Export/Import: | Wrap Cell Content:

key_id	group_id	status	k_a	k_b	hash_value
1	1000	compromised	BLOB	BLOB	-6831124328378434052
2	1001	active	BLOB	BLOB	2155982158824764557
3	1002	active	BLOB	BLOB	-8722224433160790119
4	1003	active	BLOB	BLOB	-6494578374345099477
5	1004	active	BLOB	BLOB	7716181922073026682
6	1000	active	BLOB	BLOB	-6803601381396060957
7	NULL	inactive	BLOB	BLOB	7640699948786602131
8	NULL	inactive	BLOB	BLOB	-5784801114835674760
9	NULL	inactive	BLOB	BLOB	-9033490034240231277
10	NULL	inactive	BLOB	BLOB	-7012721511857543887
11	NULL	inactive	BLOB	BLOB	-2735182576426872571
12	NULL	inactive	BLOB	BLOB	1444063446025208723

Figure 4.13 Table "keypairs" after granting a new key pair to group "group00001" when the group's key has been compromised

Scenario 5 - Access Database Scenario

Accessing an object in the database server requires a client to send a query and a piece of information to identify the group that they belong to. This information is required so the server knows which key to use for decryption. As aforementioned, the client could send the hash value of her/his key; however, I implement the case when the client sends the ID of the group that she/he belongs to (the implementation of sending the key hash value as a group identifier is kept for future work). When receiving the message from the client, Database server checks if the group has a key. If so, it gets the key and decrypts the query result then sends the result to the client.

The previous sections presented the implementation details of the database schemas and the scenarios of KCR protocol. Now, let us see how information is passed between components and how to keep the communications between these components secret.

Data Transmission

The implemented project uses Java TCP sockets. TCP stands for Transmission Control Protocol and is a protocol data transmission. A server and a client are required to establish a TCP session. A given port is set up between the client and server. The client connects to the port, and the server listens to that port for any coming packets from the client. This means that a Socket object is initiated for both of them; detailed specifications can be found in [25], [26].

Secret Communication

DES cryptosystem is used to encrypt packets sent between components. Component A initiates a connection with component B creating a secret session key then sends it to component B. The two components follow the steps of DES algorithm to encrypt and decrypt packets.

CHAPTER 5

Conclusion

Plenty of research has been conducted to improve the performance of encryption strategies used to secure large databases. However, the adoption of such research is hindered by the difficulty in providing a good key management system that prevents the workload of the re-encryption process in case a key is compromised in such databases. In this work, I propose a novel key management system named "Key Compromised Resistant protocol", referred to as KCR, that is intended to address this issue. I implement the proposed protocol and the implementation results show that KCR protocol works well.

Some features could be added to make the implemented project more efficient. Besides the tasks that KDC server is responsible for, KDC could manage adding and deleting groups. In this case, it is not necessary that groups be predefined. The same could be applied to members. Another feature might be generating public and private keys automatically in case Keys server is out of key pairs. Accordingly, all groups members are notified by email that their keys are compromised so they need to contact KDC to get their new keys.

REFERENCES

[1] B. A. Forouzan, *Data communications and networking*. New York: The McGraw-Hill Companies, 2007.

[2] B. Forouzan, *Cryptography and network security*. New York: The McGraw-Hill Companies, 2008.

[3] M. C. Murray, "Database security: What students need to know," *Journal of Information Technology Education: Innovations in Practice*, vol. 9, 2010, http://www.jite.org/documents/Vol9/JITEv9IIPp061-077Murray804.pdf.

[4] H. Kayarkar, *Classification of various security techniques in databases and their comparative analysis*, ACTA Technica Corviniensis-Bulltin of Angineering, Report ISSN 2067-3809, April-June, 2012, http://acta.fih.upt.ro/pdf/2012-2/ACTA-2012-2-25.pdf.

[5] G. I. Davida, D. L. Wells and J. B. Kam. "A Database encryption system with subkeys," *Journal of ACM Transactions on Database Systems (TODS)*, vol. 6, no. 2, pp. 312-328, June 1981.

[6] M.-S. Hwang and W.-P. Yang. " Multilevel secure database encryption with subkeys," *Data & Knowledge Engineering*, vol. 22, pp 117-131, April 1997.

[7] J. A. Cooper, *Computer and communications security: Strategies for the 1990s*. New York: McGraw-Hill, 1989.

[8] G.S. Graham and P.J. Denning, "Protection-principles and practice," *Joint Computer Conference* , vol. 40, pp. 417-429, 1972.

[9] P. A. Dwyer, G. D. Jelatis, and B. M. Thuraisingham. "Multilevel security in database management systems." *In Computers and Security*, vol. 6, no. 3, pp. 252, June 1987.

[10] W. Zhao, D. - F. Zhao, F. Gao, and G. - H. Liu. "A Cryptography index technology and method to measure information disclosure in the DAS model," *Journal of WSEAS Transactions on Information Science and Applications*, vol. 6, no. 9, pp. 1443-1452, Sept. 2009.

[11] J. Grosschadl. "The Chinese reminder theorem and its application in a high-speed RSA crypto chip," in *Proceedings of the 16th Annual Computer Security Application Conference*, 2000, pp. 384-393.

[12] S. Chen, S. Chen, H. Guo, B. Shen, and S. Jadjodia, "Efficient proxy-based Internet media distribution control and privacy protection infrastructure," in *Proceedings of the 14th IEEE International Workshop on Quality of Service*, New Haven, CT, 2006, pp. 209-218.

[13] M. Malkin, T. Wu, and D. Boneh, "Experimenting with shared generation of RSA keys," in *Proceedings of the Internet Society's 1999 Symposium on Networking and Distributed System Security (NDSS)*, 1999, pp. 43-56.

[14] D. Boneh, X. Ding, G. Tsudik, and C. M. Wong, "A Method for Fast revocation of public key certificates and security capabilities," in *Proceedings of USENIX Security*, Washington, D.C., 2001.

[15] N. Gilboa, "Two party RSA key generation," in *Proceedings of CRYPPTO*, 1999, pp. 116-129.

[16] W. Stallings, "Cryptography and network security: Principles and practice," in *Prentice Hall*, 1988.

[17] W. Diffie and M. Hellman, "New directions in cryptography," in *IEEE Transactions on Information Theory*, November 1976, vol. 22, no. 6, pp. 644-654.

[18] E. Biham and A. Shamir. "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptography*, vol. 4, pp. 3-72, Jan. 1991.

[19] M. Bellare and P. Rogaway. "Optimal asymmetric encryption - How to encrypt with RSA," in *Proceedings of the Advances in Cryptology - Eurocrypt '94*, 1994, pp. 92-111.

[20] R. L. Rivest, S. Shamir, and L. Adleman. " A method for obtaining digital signature and public-key cryptosystems," *Commun. ACM 21*, pp. 120-126, 1978.

[21] I. Mironov. (2012, Nov. 12). *Hash functions: Theory, attacks, and applications* [online]. Available: http://131.107.65.14/en-us/people/mironov/hash_survey.pdf .

[22] R. Young "DBA and developer guide to MySQL 5.6." Internet: http://dev.mysql.com/tech-resources/articles/mysql-5.6.html, 2013 [Nov. 13, 2013].

[23] G. J. Simmons. "Symmetric and asymmetric encryption," *ACM Comput. Surv.* vol. 11, no. 4, pp. 305-330, 1979.

[24] D.A. Alpern. " Factorization using the Elliptic Curve Method." Internet: http://www.alpertron.com.ar/ECM.HTM, July 2, 2013 [Nov. 30, 2013].

[25] A. Myles. "Java TCP Sockets and Swing Tutorial." Internet: http://ashishmyles.com/tutorials/tcpchat/, April 4, 2001 [Dec. 1, 2013].

[26] Q.H. Mahmoud. "Sockets programming in Java: A tutorial." Internet: http://www.javaworld.com/jw-12-1996/jw-12-sockets.html?page=1, Dec. 11, 1996 [Dec. 1, 2013]