SECURE COMMUNICATION SERVICES FOR DISTRIBUTED CONFERENCE SYSTEM

By

RAVICHANDRAN ARINGUNRAM

A THESIS PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2002
I dedicate this work to my parents, family and friends.
ACKNOWLEDGMENTS

I wish to offer my highest gratitude to Dr. Richard Newman for providing me with guidance and motivation to complete this thesis. I also thank Dr. Jonathan C. Liu and Dr. Randy Chow for serving on my thesis committee. I feel very proud to have people of such stature related to my thesis work.

I would like to express my sincere thanks to all the DCS group members. I would like to thank Vijay Manian and Aparna Kakarparsi, with whom I worked closely and shared my ideas.

Last, but not least, I wish to express my thanks to my parents, family, and friends who have always been supportive of me and always encouraged me in all my endeavors.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTERS</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview of Distributed Conference System</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Conference Control Module</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 Database and Communication Module</td>
<td>1</td>
</tr>
<tr>
<td>1.1.3 Access Control Module</td>
<td>2</td>
</tr>
<tr>
<td>1.1.4 Notification Module</td>
<td>2</td>
</tr>
<tr>
<td>1.1.5 Decision Support Module</td>
<td>2</td>
</tr>
<tr>
<td>1.1.6 Secure Communication Module</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Overview of Secure Communication Services</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Definitions</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Organization of Chapters</td>
<td>6</td>
</tr>
<tr>
<td>2 SURVEY OF RELATED WORK</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Remote Method Invocation</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1 RMI Architecture</td>
<td>7</td>
</tr>
<tr>
<td>2.1.2 Layers of RMI</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Network Security</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1 Model of Network Security</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2 Conventional Encryption System</td>
<td>10</td>
</tr>
<tr>
<td>2.2.3 Secret-Key Encryption</td>
<td>11</td>
</tr>
<tr>
<td>2.2.4 Public-Key Cryptography</td>
<td>12</td>
</tr>
<tr>
<td>2.2.5 Public Key Encryption Scenario</td>
<td>13</td>
</tr>
<tr>
<td>2.2.6 Symmetry-key vs Public-key encryption</td>
<td>13</td>
</tr>
<tr>
<td>2.2.7 Digital Signature</td>
<td>14</td>
</tr>
<tr>
<td>2.2.8 Digital Certificates</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Summary</td>
<td>16</td>
</tr>
</tbody>
</table>
3 REQUIREMENTS SPECIFICATION .............................. 17

3.1 Introduction .............................................. 17
3.2 Security Goals .......................................... 17
3.3 Processes within DCS .................................... 17
  3.3.1 Dynamic Processes .................................. 18
  3.3.2 Nearly Static Processes .............................. 18
  3.3.3 Instantiation and Termination of Servers ............. 18
  3.3.4 Creation and Destruction of Sessions ................. 18
  3.3.5 Transmission and Reception of Messages .............. 19
  3.3.6 Creation and Deletion of Sites ....................... 19
  3.3.7 Adding New Administrator to Existing Site .......... 20
  3.3.8 Creation and Deletion of Users ....................... 21
3.4 Summary ............................................... 21

4 DESIGN ...................................................... 22

4.1 Introduction .............................................. 22
4.2 Design Level Choices .................................... 22
4.3 User Authentication and Creation of a Secure Session .... 23
  4.3.1 Certificates ........................................ 25
  4.3.2 Authentication Protocol ................................ 25
  4.3.3 User Login Options .................................. 28
4.4 New Site Creation ....................................... 30
4.5 Adding a New Administrator to a Site .................... 31
4.6 Creating a New User ..................................... 31
4.7 Securing Transfer of Messages Through Network .......... 32
  4.7.1 Procedure .......................................... 33
  4.7.2 Pseudocode and Explanation ......................... 33
4.8 Summary ............................................... 37

5 IMPLEMENTATION AND TESTING ............................ 38

5.1 Implementation .......................................... 38
  5.1.1 Key Pair Generation ................................ 38
  5.1.2 Generation of Key Encryption Key .................... 39
  5.1.3 Generation of Session Key ........................... 40
  5.1.4 Digital Signature .................................. 40
  5.1.5 Digital Certificates ................................ 41
  5.1.6 Encryption and Decryption ......................... 42
  5.1.7 Conversion of Objects to Byte Array ........................ 42
  5.1.8 Objects Through Network ............................ 43
  5.1.9 Database Service ................................... 45
  5.1.10 Securing RMI Calls ................................ 46
5.2 Testing ................................................... 47
  5.2.1 Testing Generation of Key Encryption Keys ............ 48
5.2.2 Testing Signature Generation .......................... 48
5.2.3 Testing Addition of User .................................. 48
5.2.4 Testing Addition of Administrator ......................... 49
5.2.5 Testing Login Protocol ....................................... 50
5.3 Summary .................................................... 50

6 CONCLUSION AND FUTURE WORK ................................. 52

6.1 Conclusion ................................................ 52
6.2 Future Work ............................................. 53
  6.2.1 Porting Certificates to X.509 Structure .................. 53
  6.2.2 Key Migration Problem .................................. 53
  6.2.3 Considering DCS Instance as Entities .................... 54
  6.2.4 Re-Authentication ....................................... 54
6.3 Summary ................................................ 54

REFERENCES .................................................. 55

BIOGRAPHICAL SKETCH ........................................ 56
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>DCS System Architecture</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>RMI Basic Structure</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>RMI Client Server Architecture</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Layers of RMI architecture</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Conventional Encryption System</td>
<td>11</td>
</tr>
<tr>
<td>4.1</td>
<td>State Diagram for Authentication Process</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Authentication Protocol</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>User Interface</td>
<td>29</td>
</tr>
<tr>
<td>4.4</td>
<td>User Login From Other Sites</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>Code Section for SecureClientSocketFactory</td>
<td>34</td>
</tr>
<tr>
<td>4.6</td>
<td>Code Section for SecureServerSocketFactory</td>
<td>34</td>
</tr>
<tr>
<td>4.7</td>
<td>Code Section for SecureSocket</td>
<td>35</td>
</tr>
<tr>
<td>4.8</td>
<td>Code Section for EncryptedOutputStream</td>
<td>36</td>
</tr>
<tr>
<td>4.9</td>
<td>Code Section for DecryptedInputStream</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>Code Section for Key Pair Generation</td>
<td>39</td>
</tr>
<tr>
<td>5.2</td>
<td>Code Section for Key Encryption Key Generation</td>
<td>39</td>
</tr>
<tr>
<td>5.3</td>
<td>Code Section for Signature Generation</td>
<td>40</td>
</tr>
<tr>
<td>5.4</td>
<td>Code Section for Signature Verification</td>
<td>41</td>
</tr>
<tr>
<td>5.5</td>
<td>Code Section for Encryption and Decryption</td>
<td>43</td>
</tr>
<tr>
<td>5.6</td>
<td>Code Section for Objects to Bytes and Vice-Versa</td>
<td>44</td>
</tr>
<tr>
<td>5.7</td>
<td>Code Section for Objects Through Network</td>
<td>45</td>
</tr>
<tr>
<td>5.8</td>
<td>Code Section for Inserting Objects in Tables</td>
<td>46</td>
</tr>
</tbody>
</table>
Network Security has been the mantra of the last decade with the growth of the Internet. Security has evolved from the most basic forms like photo identification, signatures, and passwords to the most advanced levels like cryptography, digital signatures, certificates and bio-identification, which includes voice recognition, retina matches, etc.

Distributed Conference System v.2 (DCS) is being developed in the University of Florida under the guidance of Dr. Richard Newman. DCS supports distributed collaborative work in which important information may be transferred over the network by users. Security issues become important as more people start using the system from different domains. Basic security goals like confidentiality, authenticity and integrity had to be incorporated into the system. This thesis deals with the design and implementation of Secure Communication Services (SCS) to provide such features.

SCS is responsible for providing cryptographic keys to the users and site servers. Also SCS performs the functions of mutual authentication between users
and sites and creating a secure session. Authentication is performed using a protocol that involves distribution of session key and private key to the user by a trusted server with which it communicates. Due to the complexity of handling asymmetric keys, the users are allowed to store the keys in a homesite. SCS provides the service of encryption and decryption of data sent through the network. All message transfers over the network are carried out by RMI calls. SCS plays the important role of encrypting and decrypting the data sent by RMI calls at the socket level. Encryption and decryption are done using the DES algorithm. This is achieved by using custom socket factories. SCS also performs housekeeping activities like generating RSA key pairs for users and sites, creating digital signatures, certificates, and securely adding sites and new administrators to sites in a DCS instance.

The above mentioned functionalities have been implemented and are presented as a separate module to DCS v.2. Thus SCS provides the important security features needed for distributed collaborative work.
1.1 Overview of Distributed Conference System

Distributed computing forms the backbone of the architecture used for sharing data. A conference system plays an important role in the life of people who want to share data across the globe. Distributed Conference System (DCS) grew out of the desire to incorporate joint control over shared environments. The Distributed Conference System is a package that provides infrastructure for real-time co-operative work. DCS is an example of distributed system designed to support conferencing over wide area network (WAN). This system allows geographically separate users to collaborate in the preparation of documents, reports, software etc. The various modules of DCS are described briefly in the following subsections.

1.1.1 Conference Control Module

This module is responsible for the creation process or the boot up process of DCS. It’s duties include initialization of other modules, creating conferences, adding users, merging conferences, deleting conferences, removing users, providing GUI, etc.

1.1.2 Database and Communication Module

This module [1] provides the database services for DCS. A distributed database is created and maintained by this module. Integrity and consistency of distributed databases are maintained by this module. Reliable communication is provided by the reliable causal order multicast. All commands from one host are executed in order at all sites participating in that conference.
1.1.3 Access Control Module

This module deals with the access control issues for the conference. All subjects in DCS are bound to certain roles. Each role has its own capabilities. A subject can perform an action on an object only if the role to which he is bound permits him to do so. A subject can bind to a number of roles, but only one at a time. This module maintains an access control matrix to facilitate access control decisions.

1.1.4 Notification Module

Notification mechanisms [2] are used to inform agents about different events happening in the system; these are occurrences of observable activities. This module is responsible for notifying users of events such as user logs in, log out, joins a conference, leaves conference, etc. Various means like email, zwrite, mailbox are used to notify the users.

1.1.5 Decision Support Module

Decision Support System(DSS) facilitates the solutions of problems by a group of people who have joint responsibility of reaching the decision. A voting mechanism is one of the tools of DSS. It provides online voting and has an automated vote collection mechanism. The system provides users with different voting methods. The system also provides an interface that allows services to initiate decision process for conference control activities.

1.1.6 Secure Communication Module

Secure communication services is responsible for authentication of users, creation of keys, certificates and providing secure communication channel. This thesis concentrates on the secure communication services.

All these modules interact and communicate with each other as show in the figure below. In the current version of DCS all the services are implemented
Figure 1.1: DCS System Architecture
using Java, which helps in portability and integration of the modules on various platforms.

1.2 Overview of Secure Communication Services

The need for information security has undergone tremendous changes during the last few decades. Major advance was made when networks were introduced and data transmitted through the networks. Network security measures were need to protect the data during transmission. The Secure Communication Services provide computer and network security for the users of DCS. Computer and network security research and development have focused on three or four general security services that are need for protecting data. The security services are listed below

- **Confidentiality:** Ensures that the information in a computer system and transmitted information are accessible for reading only by authorized parties.

- **Authentication:** Ensures that the origin of a message is correctly identified, with an assurance that the identity is not false.

- **Integrity:** Ensures that only authorized parties are able to modify computer system assets and transmitted information.

- **Nonrepudiation:** Requires that neither the sender nor the receiver of a message be able to deny the transmission.

SCS is responsible for providing all these features. To enforce the important security properties (such as confidentiality, authentication and integrity) of computer and network systems, it is necessary to establish the identity of the entities of the system. The entities of the system can be users and hosts or servers. The identity of the entities should be done reliably, that is, it should be authenticated. DCS poses a greater challenge since entities communicate with messages and hence both must establish their identity and maintain confidentiality. In DCS identity is first established and usage of cryptography provides the confidentiality and integrity. The users of DCS roam about and this poses an even bigger challenge to meet the security goals.
The Distributed Conference System version two is intended to provide distributive collaboration across multiple domains. Hence a strong authentication process is necessary in order to provide the basic security features and properties. Moreover the security services should be able to handle roaming users. This makes it more challenging as the users do not keep the keys with them when they roam. Hence there should be a home domain where the users can store their keys.

In order to provide the authentication, confidentiality and integrity features an infrastructure should be maintained that handles key creation and maintenance. Also creation and maintenance of certificates is needed. The infrastructure should be created and updated whenever users are added or sites are added or site administrators are added. Entity identity is established and authenticated using this infrastructure.

1.3 Definitions

This section gives a brief definition of certain terms that are used frequently in the coming chapters.

- **Entity** A subject, object or application. An entity can only be one of these at any given moment.

- **Subject** The class of active entities that can act on objects. A Subject is always bound to a role.

- **Objects** The class of passive entities that can be acted upon by subjects. Objects can contain, and can be contained by, other objects.

- **Conference** A tuple consisting of subjects, objects, applications, roles and allowable subject-role bindings.

- **Keys** A random number generated by a given algorithm which has a property of encrypting and decrypting data.

- **Session key** A shared symmetric key between two entities used for encrypting and decrypting messages for a particular session.

- **Key Pair** A pair of public and private key for an entity.
• **Public Key** A part of a key pair that is made public to everyone

• **Private Key** A part of a key pair that is kept secret by the owner of the key pair

• **Passphrase** A string which is kept secret by an entity and is used to authenticate a person.

• **Digital Signature** Produces the same effect as a real signature; it is a mark that only a sender can make, but others can easily recognize as belonging to the sender. Just like a real signature, a digital signature is used to confirm agreement to a message.

• **RMI** Remote Method Invocation.

### 1.4 Organization of Chapters

The next chapter deals with the previous work done in the related areas. It discusses about the encryption, keys, certificates and RMI. The third chapter lists the requirements of the Secure Communications Services module. It provides with an exhaustive list of requirements. The fourth chapters deals with the detailed design of the SCS module. It gives a vivid description of the authetication protocol and creation of secure channel for communication. The fifth chapter deals with the implementation details of the module. It also lists the various test cases which were give during the testing phase. The final chapter summarizes and concludes and it also discusses the future work which can be done in this area.
CHAPTER 2
SURVEY OF RELATED WORK

This chapter gives a list and describes the relevant topics and background work done before designing SCS module. Initial section deals with the detailed description of RMI. The latter sections deals with important topics in network security and gives a brief description of each topic. With the study of these topics and relevant information, the requirements and design of SCS module is discussed in the following chapters.

2.1 Remote Method Invocation

Remote Method Invocation (RMI) was introduced to elevate network programming to a higher plane and to provide abstraction to the users. The primary goal of RMI is to allow programmers to develop distributed programs with the same syntax and semantics used for non-distributive programs. In order to achieve this, Java objects and classes on a single machine are mapped to a model were classes and objects work on a distributed environment.

2.1.1 RMI Architecture

RMI architecture [3] describes how objects are used in distributed environment, parameter passing, exceptions, memory management and how return values are managed. 

Interfaces: RMI uses an important property: the definition of behaviour and the implementation of that behaviour are separate concepts. The code which defines the role of a function and the code that implements the role are separate and are run on different machines. This fits correctly in the architecture of distributed environment. Definition of a remote service is coded using Java interface. The implementation of the service is coded in a class. This can be seen in Fig 2.1 Java
interface does not contain any executable code. There are two classes that implement the interfaces. One class gives the implementation of the functionality of the remote service and it is run on the server side. The other class acts as a proxy for the service and it runs in the client side. This is shown Fig 2.2. The client program just calls methods on the proxy object and RMI passes on this request to the remote machine and gives it to the implementation part. Return values are sent back to the proxy and then they are given back to the client program that made the call initially.
2.1.2 Layers of RMI

RMI is built from three layers. The first layer is the Stub and Skeleton layer, the second layer is the remote reference layer and the third layer is the transport layer. This is shown in Fig. 2.3.

**Stub and skeleton layer.** A stub and a skeleton class if generated whenever an RMI application is developed. The stub class plays the role of the proxy and the remote service implementation class plays the role of the real object. The skeleton class plays the role of a helper class for the RMI. The skeleton knows how to communicate with the stub. The skeleton carries on the conversation with the stub.

**Remote reference layer.** The remote reference layer is responsible for the RMI connection. This layer provides an object that acts as a link to the remote service implementation. RMI provides one way for clients to connect to the remote service. A unicast point-to-point connection is made. The remote service is instantiated and is registered at a well known agent. A well known agent is rmiregistry. This should be done before the client makes a call to the remote service.

**Transport layer.** The transport layer is responsible for providing connection between different machines. All the connections established by this layer are stream-based network connections that use TCP/IP.
2.2 Network Security

2.2.1 Model of Network Security

In this section a model of a network security system is explained in general terms[4]. Each aspect of this model is discussed in detail in the following sections.

Security is needed where it is desirable to protect the information transmission from an intruder or an outsider who may present a threat to confidentiality, authenticity, etc. All security models have two components:

1. A security related transformation on the data to be sent through the network.

2. A secret information shared by the communicating entities. This could be a secret symmetric key or a secret function.

Transformation could be achieved by encryption or addition of some code based on the sender for identification purposes. The secret information could be a secret symmetric key or a secret function. In some cases a trusted third party might be needed for mediation. The third party may be responsible for distributing the secret information or may arbitrate disputes between the entities. The general model has four steps in designing the security service:

1. design an algorithm for doing transformation of the data;

2. generate a secret information to be used;

3. develop methods for the distribution and sharing of the secret information;

4. determine a protocol to be used by the two entities that makes use of the algorithm and the secret information to achieve a security service.

2.2.2 Conventional Encryption System

The above figure illustrates the conventional encryption model. The original message is referred to as plaintext. Plaintext is converted into what appears to be random nonsense and this is known as ciphertext. The encryption process is done using an algorithm and a key. The key is a value independent of the plaintext. The
algorithm will produce a different output depending on the specific key being used at that time.

Once the cipher text is formed it is transmitted through the network. On the destination side the cipher text is transformed back to the original plaintext by using a decryption algorithm and the same key that was used for encryption.

Cryptographic systems are classified under three different contexts

1. *Type of transformation used for encryption or transformation*. All encryption algorithms can be divided into two categories: substitution, in which each element in the plaintext is mapped to another element and transposition, in which elements in the plaintext are re-arranged.

2. *The number of keys used*. If both sender and the receiver use the same key, then the system is referred to as symmetric, single key or secret key system or a conventional encryption system. If the sender and the receiver each use a different key the system is referred to as asymmetric, two key or public-key encryption system.

3. *Processing of Plaintext*. A block cipher processes the input one block of elements at a time. A stream cipher processes the input elements continuously.

### 2.2.3 Secret-Key Encryption

Secret-key encryption or symmetric-key encryption, or conventional cryptography uses one key for both encryption and decryption process. Data Encryption Standard (DES), is an example symmetric encryption.
Symmetric key cryptography [5] is very straightforward. A key is used to encrypt the data and the same key is used to decrypt the data on the receiver’s end. The security depends on the strength of the key and the strength of the encryption algorithm.

Since the same key is used in both the sender’s and receiver’s side, distributing this key becomes a major problem in symmetric encryption systems. The question is how do you keep the key safe with the sender and transmit it securely to the receiver. If there is a way by which keys can be transmitted securely then the same method can be used to transmit data instead of using a key and the encryption algorithm. If the key is shared between more than one person, then there is no way to establish the authenticity of the originator. Any person who knows the secret key can generate the message using that key. These problems led to the development of Public-key cryptography.

2.2.4 Public-Key Cryptography

Public-key cryptography was introduced in the late 70’s. A pair of keys, is used to encrypt and decrypt messages. Each pair of keys designated to an entity consists of a public key and a private key. The public key is made public by distributing to all the entities of the system. The private key is kept as a secret and hence only the owner of the private key has knowledge of it. The public and private key of a particular pair uniquely complement each other.

Public key cryptography is useful and different because of the property that data encrypted with a private key can only be decrypted with the corresponding public key. Likewise, data encrypted by the public key can only be decrypted by the corresponding private key. Another important feature of public key cryptography is that all the communication involves only the public keys and no private key is ever shared. The size of the key usually gives the strength of the key. The larger the key size, more is the security.
The public key is derived mathematically from the private key. The keys are mathematically related to each other. RSA (Rivest, Shamir, Adleman) is one of the most widely used public key encryption methods. The key pair is derived using the concepts of big prime numbers and factorization concepts. Even though the two keys are mathematically dependent it is extremely difficult or computationally infeasible to derive the private key from the public key as far as we know.

2.2.5 Public Key Encryption Scenario

Let us consider three entities A, B and C of a particular system. Each entity has a key key pair which is of the form $K_x$ and $K_x^{-1}$ where $K_x$ is the public key and $K_x^{-1}$ is the private key of entity X. Suppose A and B want to have a secret session between them and they do not want C to know about their conversation. This can be achieved in the following way

- A will send the message to B encrypted using B’s public key.
  Let M1 be the message. The cipher text is $\{M1\}_{K_B}$

- The cipher text is sent to B. When B receives the message then he will decrypt the ciphertext using his private key.
  $\{\{M1\}_{K_B}\}_{K_B^{-1}} = M1$
  This according to the property that which states that data encrypted by public key can be decrypted only by the corresponding private key. Thus B is able to retrieve the plaintext from the message it got from A.

- The above explanation shows that A can send an encrypted message to B and B can retrieve the original message successfully. It has to be proven that C cannot listen to A and B’s conversation. Suppose C gets a copy of the ciphertext $\{M1\}_{K_B}$. C cannot decipher anything from this ciphertext. He has to have a key to decrypt this message. The keys which C has are his own public and private keys, A’s public key and B’s public key. The message can be decrypted only by B’s private key. Thus with the keys C has he cannot retrieve the original message.

2.2.6 Symmetry-key vs Public-key encryption

Both conventional and public-key encryption systems have their pros and cons. Conventional encryption is very simple but key distribution is the major disadvantage. Key could be exchanged between two entities in person, which solves
the key distribution problem. This is can be done because, key distribution is not a
time critical problem. But, the same thing cannot applied to messages, as messages
has to reach the destination at the right time. Public key crypto systems avoid the
issue of key distribution but this comes with a price. The problem associated with
this system, is the association of keys with entities in the system. The processing
power needed to encrypt and decrypt messages is very high when compared to
conventional crypto systems.

2.2.7 Digital Signature

Digital signatures are similar to hand signatures. Digital signature au-
thenticate a message and provide the feature of non-repudiation. Digital signature(ideally) prevents the sender from later denying that he did not send the
message. Digital signatures are obtained using public key cryptography. A digital
signature can be described as follows. Consider two entities A and B and let each
one have their own key pair. If A wants to send a message to B and sign it, he will
do the following:

- Let the message be M. A encrypts the message M using his private key and it
  is represented as $\{M\}_{K_A^{-1}}$. This is the digital signature.

- A sends the original message along with the signature to B.

- B decrypts the signature using A’s public key.

- B checks whether the original message matches with the message in the
  signature

- If they match then it is proved that A has sent the message and it proves the
  identity of the sender. Only A could have generated a corresponding
  signature to the message M because only A has his private key. Hence
  message authentication is done.

- If A later denies that he did not send message M to B, then B can fight back
  saying that since only A has access to his private key only A could have
generated a message and the corresponding signature. Hence digital
  signatures provide the feature of non-repudiation.
Since a digital signature usually uses hashing on the message, the signature generated would be different for each message. Hence it is not possible to use someone else’s signature and attach it to the message. If this is done then when the signature is original the messages will not match.

Digital signatures are superior to handwritten signature in certain ways, that it cannot be counterfeited and also in that it applies to the whole message. In the case of handwritten signature the content of the message can be later changed even after the signature has been made. But digital signature does not imply that, the person who signed the document was physically present when signature was done.

The security of the digital signature is dependent on the security of the private key. If the private key falls in the hands of a malicious person then digital signatures can become a nightmare. Another key issue that could limit the practical use of digital signature is that the public keys for a person upon which another relies may have been falsely created. This is called man-in-the-middle attack. One may create a private key and circulate the corresponding public key as belonging to someone else. If another person relies upon this false public key then he might actually end up communicating with the wrong person. To avoid this digital certificates are needed.

2.2.8 Digital Certificates

Public key cryptosystems can be effectively put into practical use only if it made sure that public keys are distributed securely. The sender who encrypts using public key or reciever who decrypts using the public key (in case of digital signatures) should be sure that they are using the public key that corresponds to the correct person. Exchange of public keys is an important issue and the way it is done is through digital certificates.

A digital certificate is provided by a third party and it says that a public key belongs to this corresponding owner. A digital certificate contains the public key,
name of the person that the key belongs to, and a digital signature. The certificate may also include the certifier, certificate number, etc. The purpose of the digital certificate is to give an assurance to a person by a third party. The third party vouches for the authenticity of an entity. Usually the third party who does this is called as a Certification Authority (CA). Another important thing about digital certificates is that they allow you to establish the identity of the person with who you are dealing.

2.3 Summary

The study of relevant materials like RMI and topics in network security is very important for the work done in this thesis. The three layers of RMI namely, stub and skeleton layer, Remote reference layer and the transport layer were explained in detail. Also a detailed explanation of how sockets are use was give. A general model of network security was explained. Digital signatures, certificates, conventional crypto systems and public key crypto systems were discussed in detail. With these topics in mind the requisments and design of the SCS module was done. SCS module requires an extensive use of the study done in this chapter.
CHAPTER 3
REQUIREMENTS SPECIFICATION

3.1 Introduction

SCS is a subsystem that is responsible for providing the features necessary, for the secure operation of DCS. The initial part of this chapter deals with the security goals and policies of DCS. The latter part describes the processes in DCS that need to be considered for dealing with security requirements and the possible mechanisms used to attain the specified security goals. Refer to Newman and Greenwald [6] for the complete set of requirements for the entire DCS system.

3.2 Security Goals

SCS has to provide mechanisms to achieve the security goals. These features must be added to the existing DCS by SCS, as an individual subsystem and by interacting with the existing subsystems of DCS. The important high level security goals can be listed as follows.

- Identify and authenticate entities of DCS, namely users and sites.
- Message transfers in DCS must be done confidentially and means to identify the senders and recipients must exist.

The following sections gives a list of processes occurring within DCS, the necessary features that are to be added to the existing processes to meet the security goals and the possible mechanisms to attain the specified requirements.

3.3 Processes within DCS

All activities going on within DCS fall in one of the two categories namely dynamic processes and nearly static processes.
3.3.1 Dynamic Processes

These are processes that takes place very often. These processes change the security state of the system. These processes are listed below.

1. Instantiation and termination of servers.
2. Creation and destruction of sessions.
3. Transmission and reception of messages.

3.3.2 Nearly Static Processes

These processes occur less frequently and do not affect the state of the system. They cause some changes only to the database table. List of processes to be considered are listed below.

- Creation and deletion of sites.
- Adding new administrator to existing site.
- Creation and deletion of users.

The following three sections discusses about the security requirements for dynamic processes.

3.3.3 Instantiation and Termination of Servers

Instantiation and termination of servers must be done only by administrators. The administrator and the server must be able to authenticate to other existing servers in the system. The new server must be able to join into the existing system by authenticating itself and must be in a position to authenticate others. The new server must be able to access tables and send messages confidentially and with integrity.

3.3.4 Creation and Destruction of Sessions

A session is created before any communication between the users and the sites. To meet the security goals, users and sites must be identified and authenticated. SCS is responsible to provide the features necessary to strongly authenticate parties
at both ends and to start a secure session. This may require a client process to act on behalf of the user and a protocol to establish mutual authenticity. Once mutual authenticity is established the user and the site must be able to communicate securely. SCS is responsible to protect the confidentiality and the integrity of messages after a secure session is established.

3.3.5 Transmission and Reception of Messages

A DCS instance has multiple sites and message or information needs to be transferred from one site to another over the network. One example of such information is propagation of updates to the tables maintained by the distributed database service. Each table has a replica in every site and any update to an entry in the table has to be reflected in all the copies so that the database is consistent. This propagation is done by RMI calls by the database service. Likewise any communication of sites over the network is done by RMI calls. To incorporate confidentiality and integrity in message transfers is one of the security goals. SCS takes the responsibility of providing confidentiality and integrity in message passing through RMI calls. To provide this feature all RMI calls must be made secure. Encryption and decryption of data sent through RMI calls is one of the ways to meet the security goals.

The following three sections discusses the security requirements for nearly static processes.

3.3.6 Creation and Deletion of Sites

New sites are added to the system by the CCS in the current system. Each site is uniquely identified by the site id. As in the case of users, sites also require a stronger identification scheme. Associating sites with key pair is one of the ways to achieve stronger identification. A new site must be created by an administrator. The administrator must be an authorized person and so he should be a valid DCS
user. When the admin creates the site, he must also run the utility, that creates the key pair for the new site.

Once the site keys are formed there must be a way to make the new site to join the DCS instance. When the new site enters the DCS instance the transition must be smooth. In no case there must be a any halt in the normal operation of the rest of the DCS instance. The certificates of the new site must be formed. The certificate must be distributed to the rest of the sites in the DCS instance so that they can identify and trust the new site. Also the certificates of the other sites must reach the new site eventually. These communications in no way should hinder the normal operation of DCS. A site can be deleted from the system. Only administrators must be allowed to delete sites from the system. Whenever a site is deleted SCS is responsible for destroying the certificates and key associated with site.

3.3.7 Adding New Administrator to Existing Site

A site may require a new administrator for different purposes. Hence there must be provision to create new administrators for a site and strongly associate the new admin with the site. There may be situations in which a site might require more than one administrator. Hence options must be available to add a new administrator and also make multiple administrators options open. When a new admin is added, he must be authorized first. Utilities must be available to check the new person’s credibility, by checking his certificates and making sure that they are not corrupted and are valid. When multiple administrators are not needed, the old admin must be removed, once the new admin is included into the system. When multiple administrators are allowed then all administrators must have equal authority over the site. Each of them must be able to perform all the operations the other administrator can perform. Any administrator must be able to start
or terminate servers in such a way that it can authenticate itself to other servers within its site and to other sites.

3.3.8 Creation and Deletion of Users

Users are added and deleted by the CCS. Users are identified by a unique DCSID and a passphrase in the current system. A stronger identification scheme is required. One of the ways of achieving the requirement could be done by associating every user in DCS with a unique key pair. SCS is responsible creating the key pair and associating the key pair with a corresponding user. Since keys are long and difficult to remember, there must be ways to store the keys and retrieve the keys in a secure fashion. Whenever a user is removed from the system SCS must take the responsibility of destroying the key pair associated with the user. User must be able to login at any DCS site in the instance, and desirable to be able to login from any location with DCS client.

3.4 Summary

The main security goals have been listed and the processes that are to be modified to meet the security requirements are identified. The goal was to identify and authenticate entities of the system and to make all message passing confidential. Processes were classified as dynamic processes and nearly static processes. Each process was analysed and the features to be added were recognized. The requirements that are to be met by occurrence of each event have been discussed in detail and possible ways to achieve the requirements have also been pointed out in this chapter.
CHAPTER 4
DESIGN

4.1 Introduction

In this chapter the detailed design aspects of the SCS module are discussed. A detailed description of the user login protocol process is explained. This chapter also describes the structure of the necessary certificates, method of storage and the usage of certificates in the protocol. Latter sections explain the steps to be taken in terms of security features during new user creation, new site creation and new admin creation. The last section of the chapter deals with the steps involved in securing RMI calls which in effect provides a secure DCS channel.

4.2 Design Level Choices

Design level choices were made to meet the requirements and the structure of the existing system in mind. The authentication protocol was designed in order to give a lot of flexibility to the user. The design of the protocol was made in such a way that a roaming user need not have to remember a long key pair. Certificates were designed to keep the process of authentication simple, and with minimum number of handshakes. Customizing socket factories of RMI made message passing confidential. Certificates are formed for both users and sites. Hence the below mentioned protocol and securing RMI calls are necessary to create a secure session between a user and a site, instead of using SSL. SSL provides authentication between servers and not between users and servers. In the case of SSL, a malicious user can use the system after the servers are authenticated. PKI could not be used due to its inherent nature. PKI used a hierarchical structure which was not suitable to the DCS architecture. Also PKI uses revocation lists which is not applicable to DCS certificates. Moreover the financial aspect of implementing PKI
proved to be another hinderence in using PKI. Using smart-cards to store keys was one of the options discussed. This increases the burden on the user, as he has to carry the smart-card with him wherever he goes and has to find a suitable hardware system to install the smart-card. With all these different issues in mind the most suitable protocol for DCS was designed and implemented.

4.3 User Authentication and Creation of a Secure Session

As described in the requirements chapter, the main aim of this protocol is to distribute a session key created by the server, to a client process that is starting a session. A session is started by a process of login and authentication. There are several steps involved in the mutual authentication process. It is important that mutual authentication takes place, as both the server and client should come to a conclusion that they are communicating with the right entity.

The broad outline of this process can be understood using Fig 4.1.

The following conventions are followed throughout this chapter.

- A server entity is represented as S
- A user is represented by a client process C
- The public key of an entity e is represented as $K_e$.
- The private key of a entity e is represented as $K_e^{-1}$
- A session key between two entities C and S is represented by $K_{cs}$
- A nonce number is represented as $N_x$

Before going through the steps of the protocol, it is important to understand the certificates required, the structure of the certificates and the storage mechanism of certificates.
Figure 4.1: State Diagram for Authentication Process
4.3.1 Certificates

As described in the previous chapter, certificates are formed to store and retrieve keys of the users and sites in a secure fashion. The certificates required, how they are formed and where they are stored are explained as follows.

- User provides only his DCS id and the passphrase.
- Users and sites have a key pair associated with them.
- A secret key can be generated from the user’s passphrase.
- The server S uses this secret key to encrypt the user’s key pair when the user is created or added to the system.
- The server S has two public key certificates and one private key certificate for the user C. Detailed structure of certificates can be found in Sec. 5.1.5.

- User PublicKey Certificate: \(C, K_C, S, \{N, C, K_C\}_{K_S}^{-1}\). This certificate denotes that the public key of C is \(K_C\) and it is signed by the server S. S vouches for C’s public key. This certificate is formed by the server and it is stored in the server.

- Server PublicKey Certificate: \(S, K_S, C, \{N_p, S, K_S\}_{K_C}^{-1}\). This certificate denotes that the public key of the server S is \(K_S\) and it is signed by the user C. User C vouches for the server S’s public key. This certificate is created by the user during its creation and is stored in the server.

- User PrivateKey Certificate: \(C, \{N_p, C, K_C^{-1}, K_C, S\}_{K_{pc}}\). This is a private key certificate. This certificate is formed using the secret key generated using the client’s passphrase. The secret key is used to encrypt the user’s key pair and the certificate is stored in the server. This certificate is also created when the new user is added to the system.

- Other than these the server maintains list of public key certificates for other servers. The certificate structure is as follows.

\(S', K_{S'}, S\{N, S', K_{S'}\}_{K_S}^{-1}\)

This certificate states that the public key of server \(S'\) is \(K_{S'}\). The server S vouches for server S’s public key.

4.3.2 Authentication Protocol

The protocol for authentication [7] can be explained using Fig 4.2.
Figure 4.2: Authentication Protocol
The following list gives the assumptions for the authentication protocol.

- All users and sites should have their own public and private keys.

- All users store the keys in the form of certificates in a secure manner.

- For the above reason entities encrypt the keys using a secret key, which is derived from a passphrase. The passphrase should be easy to remember but should not very common words or names of person or a thing. The encrypted key pair should be stored in the user’s homesite, at least.

- Derivation of keys from passphrase must be a simple procedure and a single passphrase should not map to different keys.

- We assume that the users do keep their passphrase a secret.

- Certificates are needed to authenticate one another.

- Hence services must be available to create, maintain and destroy certificates.

- The format of each certificate type must be pre-specified.

The steps for the authentication process are given below.

1. User logs into the client process and gives his/her DCSID and passphrase.

2. As the client process has the capability to form the secret key from the user’s passphrase, it forms the secret key and stores it locally.

3. The client process gets the id of the homesite server of the user from the global databases and forwards the DCSID and a challenge nonce to the server.

4. The client then listens for the server’s response.

5. The site first responds by giving the user’s private key certificate, which is stored in the site based on the DCSID. This certificate is nothing but the key pair of the user, encrypted by the secret key generated by the user’s passphrase.

6. The secret key generated by client from the passphrase is a symmetric key and hence the client can decrypt this certificate and retrieve the user’s key pair.

7. The server then sends its own public key certificate signed by the user’s private key. Hence the client process now gets the server’s public key.
8. Along with client’s nonce and a challenge nonce the server generates, the session key is sent by the server. This session key is signed by the server S and it is encrypted by the user’s public key $K_C$. Hence this message can only be decrypted by the corresponding private key.

9. The client process decrypts the session key using the user’s private key. Then it sends back a response to the challenge given by the server encrypted by the new session key.

10. The server checks this response and if it is a valid response then the new session is accepted and the session key is used for any further communication.

11. Thus the client and the server both come to a mutual agreement that the other person is using the new session key.

The correctness of the protocol can be theoretically explained as follows. Only if the user’s passphrase is correct, can the client process generate the symmetric key and decrypt the user’s key pair correctly. The user’s key pair is necessary to verify the server’s public key. The server’s public key and the user’s private key are required in order to decrypt the session key and to verify it. Thus the whole protocol goes on smoothly only if the passphrase provided is correct. As stated in the requirements, we assume that the user keeps his passphrase a secret. Also at the end of the protocol a challenge response to the nonce generated by the server is sent back to server using the new session key. Hence both parties are satisfied that the other person is using the fresh session key. The mathematical proof for this protocol is available in Newman et al. [7]. The proof is done using logic of authentication[8].

4.3.3 User Login Options

The user interface for the log on process is designed as follows.

The user types in his user name and passphrase. There are two possibilities for the user to log on the DCS instance.

1. User can log on through the homesite

2. User can log on through a site other than his homesite
When the user logs in from his homesite, he clicks the first radio button, which says that the current site is his homesite. When he logs in from another site, then he clicks the second radio button, which says that his homesite has to be found from the global database.

The authentication process for the user login from the homesite was described in the previous section. When the user logs in from another site the process of authentication is little different. The difference is mainly because the certificates of the user have to be brought from the user homesite. First, the current site queries the global database to get the homesite address for the user, using his DCSID.

After the homesite is located the current site contacts the user’s homesite requesting the users certificates. The homesite responds by giving the certificates required for the login process. The user trusts the homesite’s public key for the current site. The user also has to believe the authority of the homesite to say that the current site will issue a valid session key.
Figure 4.4: User Login From Other Sites

4.4 New Site Creation

A new site is added to the instance by the administrator. The administrator is a valid DCS user and has a DCSID. The admin, when he creates the site also runs the utilities that create the key pair for the new site. The new site public key certificate is formed and is sent to the admin’s homesite. The certificate is signed by the admin’s private key. The private key of administrator is obtained from the private key certificate stored in admin’s homesite. The newly formed public key certificate is as follows

\[ S', Ks', AdminDCSID, \{N, S', Ks'\}_{K_{AdminDCSID}}^{-1} \]

The homesite (S) of the administrator then broadcasts the public key to all other sites. The certificate is as follows

\[ S', Ks', S, \{N, S', Ks'\}_{K_{s}}^{-1} \]

where S is the homesite of the administrator.

The corresponding sites can verify this certificate using the admin’s public key.

Once the verification is done each site forms its own public key certificate for the new site. Thus the homesite of the administrator acts as a link between the new site and the rest of the DCS instance.
4.5 Adding a New Administrator to a Site

The following steps are to be executed by the SCS when a new administrator is added to an existing site.

1. A new admin can be added to the system only by an existing administrator. Hence the existing admin logs in to the site by providing his DCSID and his passphrase.

2. Using this passphrase the secret key is derived and it is used to decrypt the site’s key pair from the private key certificate, i.e. $K_s$ and $K'_s$ are retrieved.

3. The new administrator’s id and passphrase are obtained and a new secret key is derived from the new passphrase.

4. Using this secret key the site keys are again encrypted and a new private key certificate is formed.

5. Once the new certificate is formed the administrator database table is updated with the new admin’s id and the certificate.

6. Various options have to be dealt with here. The site can have multiple administrators. In this case the old entry for the old admin is retained in the database.

7. If only a single administrator can be present then the old admin entry in the tables are removed.

8. If the site has multiple administrators then all the administrators will have equal powers in administering the site.

4.6 Creating a New User

As described in the previous chapter, creating a new user is one of the events that is to be handled by SCS. SCS creates the key pair and certificates for identification and authentication of user. The steps to be done by SCS when a new user is created are listed below.

1. The information required by the SCS module are DCSID, username, passphrase, and the homesite id or address.

2. A fresh key pair is generated for the new user.

3. A secret key is generated that depends on the passphrase.
4. Using this secret key, the private key certificate for the user is formed. This is nothing but encrypting the key pair with the secret key generated from the passphrase.

5. The public key certificate for the homesite is then formed. A new user is created by an admin. Hence he can use his secret key to get the site’s public key. The site’s public key is signed by the new user’s private key.

6. The site forms the public key certificate of the user. The user’s public key is signed by the site’s private key.

7. All the certificates are stored in the database.

   The following steps are followed when a user needs to change his passphrase.

   1. The user enters his DCSID, old passphrase and the new passphrase.
   2. The user’s key pair is retrieved using the old passphrase.
   3. A new private key certificate is formed using the new passphrase.
   4. The new certificate is updated in the tables and the old certificates are removed.

4.7 Securing Transfer of Messages Through Network

As stated in the previous chapter all information transfer through the network in DCS takes place by RMI calls. Hence forming a secure DCS channels implies that the RMI calls should be made secure.

Securing RMI calls can be done using a Custom Socket Factory. RMI calls provide an abstraction to the user by hiding the socket level communication. The underlying protocol uses sockets for communication between the server and client. A secure channel is obtained by encrypting and decrypting the data that is sent and received by the sockets.

Installing our own RMI socket factories allows us to use a customized socket rather than a TCP socket provided by java.socket.net, which is used by RMI as default. There are five steps to create a custom RMI socket factory that produces a single type of socket[9].
1. Decide upon the type of socket to be produced.

2. Write a client-side socket factory that implements the RMIClientSocketFactory.

3. Implement the RMIClientSocketFactory createSocket() method.

4. Write a server-side socket factory that implements RMIServerSocketFactory.

5. Implement the RMIServerSocketFactory createSocket() method.

4.7.1 Procedure

Creating secure sockets involve the following steps:

1. Implement SecureClientSocketFactory class,

2. Implement SecureServerFactory class,

3. Implement SecureSocket class,

4. Implement SecureServerSocket class,

5. Implement EncryptedInputStream class,

6. Implement DecryptedOutputStream class,

7. Use the created custom socket factory.

4.7.2 Pseudocode and Explanation

SecureClientSocketFactory. This class specifies the pool of sockets for clients when it needs to start communication. Whenever a client needs a socket, a call to the createSocket() method is made. Hence in this class we need to override this method. This class will extend the RMIClientSocketFactory class. The session key is a private member of this class and it is set in the constructor. Code section is explained in Fig. 4.5.

SecureServerSocketFactory. This class extends RMIServerSocketFactory class. This specifies the pool of sockets for servers when it needs to start communication. Whenever a server needs a socket, a call to the createSocket() method is made.
public class SecureClientSocketFactory extends RMIClientSocketFactory
{
    private sessionkey;
    SecureClientSocketFactory(sessionkey)
    {
        initialize the sessionkey
    }
    public Socket createSocket()
    {
        return new SecureSocket(sessionkey)
    }
}

Figure 4.5: Code Section for SecureClientSocketFactory

public class SecureServerSocketFactory extends RMIServerSocketFactory
{
    private sessionkey;
    SecureServerSocketFactory(sessionkey)
    {
        initialize the sessionkey
    }
    public Socket createSocket()
    {
        return new SecureSocket(sessionkey)
    }
}

Figure 4.6: Code Section for SecureServerSocketFactory

Hence in this class we override this method. Fig. 4.6 explains the code level details.

SecureSocket. This class extends that standard java.net.socket class. This has methods to provide secure sockets to both client and server socket factories. The secure sockets are formed by modifying the input and output streams of the socket. Anything which goes through the streams of the sockets are encrypted. The sessionkey is obtained from the constructor and it is set as a private member of this class. Code section is shown in Fig. 4.7.
public class SecureSocket extends java.net.socket {
{
    private sessionkey;
    SecureSocket(sessionkey)
    {
        initialize sessionkey
    }
    public InputStream getInputStream()
    {
        return (new DecryptedInputStream(buffer, sessionkey);
    }
    public OutputStream getOutputStream()
    {
        return (new EncryptedOutputStream(buffer, sessionkey);
    }
}
}

Figure 4.7: Code Section for SecureSocket

EncryptedOutputStreamStream. This class extends the FilterOutputStream class. All the processing of the data which are going to pass through the network are done in this class. Every byte which is sent through the network is encrypted by the session key. Fig. 4.8 gives the code segment for encrypting data.

DecryptedInputStream. This class extends FilterInputStream. When a data is read from a socket decryption process takes place in this class. The sessionkey, since it is a symmetric key, is used to decrypt the data coming from the socket.
Fig. 4.9 describes the code details of decryption.

Using the SocketFactories. The usage of the newly created socket factory is very simple. The code for creating secure sockets should be installed in both the client and the server side. One line of code needs to be added in the constructor of the RMI server code or the class which extends the UnicastRemoteObject class. The UnicastRemoteObject class uses the default RMISocketFactories. Instead in the constructor, if we specify that the SocketFactory should be the SecureSocketFactory, then secure sockets are created whenever an RMI call is made. Thus in
public class EncryptedOutputStream extends FilterOutputStream{
    private sessionkey;
    EncryptedOutputStream(buffer, sessionkey)
    {
        initialize the session key and buffer
    }
    public message write()
    {
        Encrypt the data to be written
        write the data to outputstream
    }
}

Figure 4.8: Code Section for EncryptedOutputStream

public class DecryptedInputStream extends FilterInputStream
{
    private sessionkey;
    DecryptedInputStream(buffer, sessionkey)
    {
        initialize the session key and buffer
    }
    public message read()
    {
        Decrypt the data to be written
        return the data to inputstream
    }
}

Figure 4.9: Code Section for DecryptedInputStream
the constructor of the RMI server the following line of code is added.

\[
\text{super}(0, \text{SecureClientSocketFactory}(\text{sessionkey}), \text{SecureServerSocketFactory}(\text{sessionkey}))
\]

Once this is finished the client and server can be compiled and run as normal RMI programs. Thus this method give a very good abstraction to the user as the encryption and decryption process takes place in the underlying layers without the knowledge of the user.

4.8 Summary

The key issues of design were discussed. The assumptions made and the steps to be executed whenever a particular event occurred were described in detail. All the steps describing the design of each action, contributes towards attaining the stated security goals. The authentication protocol helps in mutual authentication of users and sites before starting a session. Mutual authentication is achieved by exchanging certificates. Once authentication is done, a secure session is created and all message transfers are made secure by using custom socket factories which encrypts messages. The necessary certificates are formed when a new user is created, or a new site is created or a new administrator is added to a site. All these certificates are updated in tables in the database. With the inclusion of all these features, DCS operations are made secure.
CHAPTER 5
IMPLEMENTATION AND TESTING

This chapter deals with the important implementation details and the testing done on the software module. The entire code was written in Java. The database used to store the details of entities is PostgreSQL database server. The first part of the chapter talks about the implementation details and the latter half describes the testing of the software.

5.1 Implementation

Implementation was started using the available Java package that was installed previously. During the implementation phase it was found that the existing package (JDK 1.3) did not have all the packages necessary to generate the symmetric keys and to support encryption/decryption. For this reason, J2SDK 1.4 version was installed in the systems. This version had the javax.crypto package. This package has the classes necessary to generate symmetric keys and methods to do encryption/decryption. A few security files had to be modified for this package to work properly. Also, it was discovered that the database module did not support insertion of objects into tables. This was a feature needed to store certificates in the tables. This feature was added to the database service and a detailed explanation about this is available in the later sections.

5.1.1 Key Pair Generation

Each and every user and site in DCS are designated a key pair that consists of a public key and a private key. The key pair is generated using the Java Application Program Interfaces (API). The API’s to generate a standard key pair is available in the java.security package. An instance of the class named KeyGeneratorPair is obtained by using the getInstance() method, which takes
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
KeyPair keys = kpg.genKeyPair();
keys.getPublic();
keys.getPrivate();

Figure 5.1: Code Section for Key Pair Generation

passphraseBytes = passphrase.getBytes();
SecureRandom sr = new SecureRandom(passphraseBytes);
KeyGenerator kg = KeyGenerator.getInstance("DES");
kg.init(sr);
skey = kg.generateKey();

Figure 5.2: Code Section for Key Encryption Key Generation

the algorithm name as a parameter. RSA algorithm is used to generate key pairs. Code section to generate key pair is shown in the following figure. generated as follows

5.1.2 Generation of Key Encryption Key

As discussed in the previous chapter, the user enters only his DCSID and the passphrase when he logs in. The passphrase is a string. A key encryption key has to be generated from the passphrase, the user enters. Also the key encryption key should be a symmetric key. In order to meet these requirements, the javax.crypto package is used. Crypto package is bundled with standard J2SDK 1.4.

The passphrase entered by the user is retrieved and is converted to an array of bytes. The byte array is passed as a seed to generate a random number. The random number is used to initialize the key seed required to generate the key encryption key. Symmetric keys are generated using the KeyGenerator class provided by Java. This class generates a key of type SecretKey which is also inherited from the base class Key. Key encryption key is generated as shown in Fig. 5.2.
Signature signature = Signature.getInstance("MD5withRSA");
signature.initSign(privateKey);
signature.update(data);
byte signdataaa[] = signature.sign();

Figure 5.3: Code Section for Signature Generation

5.1.3 Generation of Session Key

The session key is also a symmetric key. Hence the KeyGenerator class is used to generate each instance of a SecretKey. The random seed for initialization is a standard random seed provided by Java. Hence each session key is also an instance of the SecretKey class which, can be generated as discussed in the previous subsection.

5.1.4 Digital Signature

Digital signatures provide integrity of a message and also proves the authenticity of the senders. As discussed in the previous chapter, signatures are heavily used during the authentication protocol. Digital signatures are bundled in the certificate (discussed in the next subsection), and are sent and received during the process of login.

Signature class, provided by the Java API, provides the features of digital signatures. A Signature object is created using the getInstance() method, which takes an algorithm as a parameter. When this object is used to sign or verify a signature then the corresponding algorithm is used. A signature is usually made using the signing party’s private key. So, to generate a signature the Signature object is initialized using the signing party’s private key. Then the data to be signed is updated on the object followed by the call to the sign() method. This method returns a byte array that has the digital signature. Code section to generate signature is shown in Fig. 5.3

The signature generated might have to be verified by another person. The public key of a site is signed by the user and the vice versa. The user has to verify
Signature signature = Signature.getInstance("MD5withRSA");
signature.initVerify(publicKey);
signature.update(sign);
return signature.verify(signdata);

Figure 5.4: Code Section for Signature Verification

the whether the public key of the site is correct. Hence, he verifies the public key, certificate generated with his signature. The signature is verified using his public key. A Signature object is created, initialized using the public key and is updated using the signature that has to be verified. Then, the verify() method is called which returns a boolean value. True is returned if, the signature is valid and false is returned if the signature is not valid. Signature verification is shown in Fig. 5.4.

5.1.5 Digital Certificates

Design of the digital certificates is made keeping in mind the issues of portability to standard certificate formats like the X.509 structure. A study on the certificates structures was made and the design of the certificate structure is as follows,

- **Version**: Indicates the version of the certificate structure which is currently in use.

- **Sequence number**: Indicates the sequence number of the certificate.

- **Signature algorithm**: Gives the name of the algorithm used to form the signature that is present in the certificate.

- **Issuer id**: Holds the DCSID of the person who has formed or issued the certificate or signature.

- **Validity**: This is a date field that denotes how long this certificate is valid. After this date the certificate cannot be trusted.

- **Subject id**: Holds the DCSID of the entity whose public key is being signed by the issuer.

- **Subject Public Key**: Holds the raw Public key of the subject.
• *Signature:* This field is the signature. This signature is formed using the issuer’s private key. The public key of the subject is signed using the private key of the issuer. It is a byte array.

Whenever a new user is created, or a new site is added the corresponding keys are first formed. The certificate content to be signed is generated followed by generation of signature. These certificate objects are formed by populating the above mentioned fields and are stored in the database. The certificate objects are made serializable, as they need to be transported over the network during the logon process.

5.1.6 Encryption and Decryption

The entire secure communication services module is based on successful and strong encryption and decryption. Java provides good API's for encrypting and decrypting byte streams using algorithms of the user’s choice. Symmetric encryption is done using DES algorithm and asymmetric encryption is done using RSA algorithm. This is achieved using the Cipher class in the javax.crypto package. An instance of the Cipher class is obtained using the getInstance() method that takes the algorithm name as the parameter. The mode of the Cipher object is then set and also the key is initialized. The mode to be set can be ENCRYPT mode or DECRYPT mode. The key that is used during initialization is used for encryption or decryption. The byte stream to be encrypted or decrypted is passed to the doFinal() method for the encryption or decryption process. The doFinal() method returns a byte array encrypted or decrypted according to the initialization. The Fig. 5.5 gives the details of implementation.

5.1.7 Conversion of Objects to Byte Array

While constructing a signature, the update(data) method of the Signature object takes a byte array as a parameter. The data to be signed by an entity should be a byte array. Public key has to be signed by the issuer. PublicKey is an object and this needs to be converted to a byte array in order to initialize the
Encryption:

Cipher c = Cipher.getInstance("DES");
c.init(Cipher.ENCRYPT_MODE, keyencryptionkey);
byte encrypteddata = c.doFinal(data);

Decryption:

Cipher c = Cipher.getInstance("DES");
c.init(Cipher.DECRYPT_MODE, keyencryptionkey);
byte decrypteddata[] = c.doFinal(encrypteddata);

Figure 5.5: Code Section for Encryption and Decryption

Signature object. This is achieved using theByteArray streams and the ObjectOutputStream classes of Java.

When an object has to be converted to a byte array, theByteArrayOutputStream and theObjectOutputStream classes are used. A newByteArrayOutputStream object is created. This object is used as a parameter to initialize theObjectOutputStream. The object to be converted to byte array is then passed as a parameter to thewriteObject() method ofObjectOutputStream class. By doing this the object is converted to a byte array and is stored in theByteArrayOutputStream object. The actual byte array can be retrieved from the object using thetoByteArray() method.

Likewise, when the object is to be constructed from a byte array, then theByteArrayInputStream and theObjectInputStream are used. AByteArrayInputStream object is created using thegive byte array. ThisByteArrayInputStream object is used to construct anObjectInputStream object and thereadObject() method is used to get the object back from the byte array. Fig. 5.6 gives the code section.

5.1.8 Objects Through Network

This subsection deals with sending and receiving objects through the network. During the authentication protocol, the certificate objects are passed through the
Object to byte array:

```java
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(baos);
oos.writeObject(object);
byte objectbytearray[] = new SignKey(serverpublickey, baos.toByteArray());
```

Byte array to Object:

```java
ByteArrayInputStream bais = new ByteArrayInputStream(decryptedkeypair);
ObjectInputStream ois = new ObjectInputStream(bais);
EntityKeyPair userkeypair = (EntityKeyPair)ois.readObject();
```

Figure 5.6: Code Section for Objects to Bytes and Vice-Versa

Network from the server side to the user side. Generally bytes are sent through sockets. Java provides a good abstraction by allowing us to directly send objects through the network. Conversion of objects to bytes and reconstructing them at the other end is take care by JVM. In order to send an object through the network, the object must be Serializable.

To send an object through the socket, a handle to the socket output stream is required. Socket class has `getOutputStream()` method which returns the handle to the socket output stream. The handle obtained is passed to the `ObjectOutputStream` object and then the `writeObject()` method is used to write the object on to the socket output stream, which is then sent through the network.

Objects which are sent at one end have to be retrieved as objects at the other end of the network. Instead of handling bytes, Java allows programs to handle objects directly in this case also. The handle for socket input stream is obtained using the `getInputStream()` method. This handle is passed to the `ObjectInputStream` object and then the `readObject()` method is used to read the object from the network. The implementation detail is show in Fig. 5.7.
Send Objects:

```java
ObjectOutputStream os = new ObjectOutputStream(socket.getOutputStream());
os.writeObject(userprivatekeycert);
os.flush();
os.writeObject(sitepublickeycertificate);
```

Recieve Objects:

```java
InputStream is = socket.getInputStream();
ObjectInputStream ois = new ObjectInputStream(is);
Certificate sitepublickeycertificate = (Certificate)ois.readObject();
```

Figure 5.7: Code Section for Objects Through Network

5.1.9 Database Service

Secure communication service makes extensive use of the database service. When ever an entry is to be made in the database while creating a user or creating a site, then a call is made to the database service module. The database service module creates a connection to the database name specified and executes the query given and returns the result set. Database module did not support the insertion of objects into tables. SCS needs certificates to be stored in tables of the database. This functionality was added to the database module in order to store certificates in the tables. Two methods are added to the database module, which basically stores the objects in tables and retrieves objects from tables.

Postgre SQL database supports object storage [10] in tables and objects stored are made persistent. Postgre SQL provides two distinct ways to store binary data. Binary data can be stored in a table using PostgreSQL’s binary data type bytea, or by using the Large Object feature which stores binary data in a separate table in a special format and refers to that table by storing a value of type OID in the table. Using the LargeObject method is more suitable for storing certificates as they are very large. To use large object functionality the LargeObject API provided by PostgreSQL is used. Access to LargeObjects must be made within a transaction.
// All LargeObject API calls must be within a transaction
boolean oldstate = db.getAutoCommit();
db.setAutoCommit(false);
// Get the Large Object Manager to perform operations with
LargeObjectManager lobj = ((org.postgresql.Connection)db).getLargeObjectAPI();
// Create a new Large object
oid = lobj.create(LargeObjectManager.READ | LargeObjectManager.WRITE);
// Open the large object for write
LargeObject obj = lobj.open(oid, LargeObjectManager.WRITE);
// Convert the certificate object to a byte array
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(baos);
oos.writeObject(cert);
byte objectByte[] = baos.toByteArray();
// Copy the data from the object to the large object
obj.write(objectByte,0,objectByte.length);
// Close the large object and the transaction.
obj.close();
db.setAutoCommit(oldstate);
// Command is the sql INSERT statement in which '?' is present in the place of the certificate value
PreparedStatement ps = db.prepareStatement(command);
ps.setInt(1,oid);
ps.executeUpdate();

Figure 5.8: Code Section for Inserting Objects in Tables

transaction must be opened and this is done using the setAutoCommit() method with the input parameter of false. Fig 5.8 gives the code section to store objects.

Large Object method is used to retrieve an object stored in the database. The oid is used to reference the object stored in the special table. Fig. 5.9 gives the code level details.

5.1.10 Securing RMI Calls

Implementation of this module, was started by writing a small application using RMI. Once the RMI application was developed, code for the socket factories were written. The encryption/decryption part was left empty so as to just check whether socket factories were used properly. Once this was verified, the session key was passed in the constructors and code to do the encryption/decryption was introduced. There were few specific problems encountered at this stage of the development process. DES algorithm was used for symmetric encryption/decryption.
// Open a transaction for accessing large objects
db.setAutoCommit(false);

// Get the large object manager to perform operations with
LargeObjectManager lobj = ((org.postgresql.Connection)db).getLargeObjectAPI();
// Prepare SQL query and execute
PreparedStatement ps = db.prepareStatement(command);
ResultSet rs = ps.executeQuery();
while(rs.next())
{
    // Get the oid reference from the result set
    oid = rs.getInt(1);
    // Open the large object for reading
    LargeObject obj = lobj.open(oid, LargeObjectManager.READ);
    byte buf[] = new byte[obj.size()]
    // Retrieve the object using oid and store as byte array
    obj.read(buf,0, obj.size());
    obj.close();
    // Convert byte array into object
    ByteArrayInputStream bais = new ByteArrayInputStream(buf,0,buf.length);
    ObjectInputStream ois = new ObjectInputStream(bais)
    return ois.readObject();
}

Figure 5.9: Code Section for Retrieving Objects from Tables

Initially an attempt was made to encrypt every single byte of data one by one. This process failed as DES algorithm expected multiples of 8 byte blocks to be encrypted/decrypted. One simple solution arrived at initially was to do an XOR encryption using the session key. This worked fine as each byte could be encrypted individually. Later multiples of 8 bytes were given to the method to read and write.

5.2 Testing

Software engineering policies were used throughout the development of the SCS module. The spiral approach was used, which gave the opportunity to go back and modify things. A step-by-step procedure was followed. Weekly meetings were held with the entire team and brain storming sessions helped to come up with the requirements of the SCS module. The design of the module was open to criticism of the DCS group members which helped in correcting the faults in the original design.
Different test cases were given for different scenarios. Initially test cases were designed to see if all the functionalities are present and then test cases were developed to find bugs and to break the programs.

5.2.1 Testing Generation of Key Encryption Keys

Key encryption keys are derived from a passphrase. The passphrase is nothing but a ASCII string. Every user has a passphrase, and hence the key encryption key generated using this passphrase should also be unique. In order to check this multiple test cases were given to the program which generates the key encryption key. A auxiliary program is used to display the key generated in base64 format. The program was run by giving different ASCII strings as input to the program and making sure that the same key is not generated. Also same passphrase was given as input multiple times and it was made sure that same key is generated. This situation occurs when the same user logs in again from another machine. His key encryption key must be the same whenever he logs in unless he changes his passphrase.

5.2.2 Testing Signature Generation

This module is tested using a key pair and a byte data stream. The object is initialized using the key pair. The signature is formed using the private key of this key pair. The testing is done by checking whether a byte stream is returned after the signature is formed. Also, the signature formed is correct only if it can be verified. The signature is verified using the public key. The signature byte and the public key are given for verification. The verification method returns a true if the signature is correct, else false. Hence for testing purposes the verification method was used to check the correctness of the signature generation module.

5.2.3 Testing Addition of User

This module was tested by simulating the situation of adding a new user to the system. In the actual system, this is taken care by the Conference Control
System and the necessary parameters are passed on to the SCS module to take care of security features. Hence the program was run by giving these parameters. The DCSID, user name, passphrase, adminid, admin passphrase and homesite id are given as parameters to test this functionality. The process is tested by keeping track of the following steps.

- Key pair should be generated for the new user.
- Key encryption key should be generated using the passphrase.
- Key pair should be encrypted using the key encryption key.
- Public key certificates should be formed.

All the above listed items are dumped on the screen as and when they are created. All the details of user and the certificates are stored in tables in the database. The database is checked to see whether a new entry has been made for the new user in the corresponding tables. The certificates of the user are stored as objects. Testing scenario for user changing passphrase was also designed. Once the user changed his passphrase new certificates were formed and corresponding tables were updated.

5.2.4 Testing Addition of Administrator

Simulation of adding an administrator was made by running the program with the corresponding parameters. These include siteid for which the admin is added, old administrator’s DCSID, old admin’s passphrase, new administrator’s DCSID, new admin’s passphrase and option. This option could be *retain old admin* or *delete old admin*. This module creates a new entry in the administrator information table with the new encrypted version of the key pair of the site. Testing was performed by giving incorrect old admin DCSID. Also new administrator should be a valid DCS member and hence test cases was carried on for these situations also.
5.2.5 Testing Login Protocol

To test the login protocol a server program was needed. For unit testing purposes a multi-threaded server program was written to receive requests from a client process and give the requests to a request handler by starting a new thread. This program served as the DCS server for authentication protocol testing cases.

The entire protocol was developed in stages with testing at each stage. The client program receives the DCSID and passphrase and generates the key encryption key. The DCSID and nonce are sent to the server. Screen dumps were used to check whether the server receives this properly. The second stage developed was in the server side which sends certificates of the user to the client. Testing was done to check whether the client receives the correct certificates and whether it was able to decrypt the certificates properly. The third stage developed was the response by the client back to the server and nonce values were checked to see the proper working of the program. Correct DCSID and passphrase was given to test whether a user was able to log in successfully. The protocol was also tested for unsuccessful logins. Improper combination of DCSID and passphrase were given to the client program for this purpose. Also multiple users were allowed to log in simultaneously to test the scalability and the robustness of the protocol.

5.3 Summary

This chapter described the important aspects of the implementation and testing done for the SCS module. Key implementational issues that may be useful for future workers of DCS are dealt in detail. The test cases gives the opportunity for readers to understand the working of the SCS module. Code sections for generating key pairs, key encryption keys, signatures and encryption/decryption methods were discussed. The packages used to implement these features were also listed out. JSDK 1.4 was installed and used. The javax.crypto package and java.security package were used extensively. RSA algorithms were used to
generate key pairs. DES algorithm was used for generating symmetric keys and for symmetric encryption. Signatures were stored in the form of certificates. These certificates were exchanged during the authentication protocol. Addition to the database service module has been made. It is now feasible to insert and retrieve objects from the database. Objects are stored in the form of LargeObject type. The datatype used to store objects was oid. Test cases helped in testing the functionality and to correct the bugs in the programs.
The objective of this thesis was to make all the operations of DCS secure. The security goals of the system were listed. These include, identification and authentication of entities of DCS and to make all the message transfers with DCS confidential. The processes within DCS were identified that required modifications to achieve the security goals. The requirements for modifications were analyzed during the requirements analysis phase. A detailed design of all the features to be added were made with all the requirements in mind. Implementation and testing followed the design process.

By providing an effective authentication protocol for login, creating a secure DCS channel and securely adding sites, users and administrators to a DCS instance, the above mentioned security requirements have been achieved. These features have been successfully implemented. The SCS module hence has become an important module in the DCS architecture.

The authentication protocol uses cryptographic keys successfully to maintain a confidential and mutually authenticated key exchange. The result of the protocol is that, the entities believe that the session key is good and that the other entity has the same belief. Implementation of this protocol has resulted in entities authenticating themselves and are able to engage in a secure session without facing the challenge of memorizing an asymmetric key.

The secure DCS channel ensures that message integrity is maintained. Also by providing security for RMI calls DCS users and developers are still able to enjoy the level of abstraction provided by RMI. Securing RMI provides abstraction for
users and developers, enabling them to proceed with their work without thinking about security aspects.

By implementing protocols to add sites, add users and adding administrators, the system moves from one state to another without causing any hinderence to the normal user.

6.2 Future Work

6.2.1 Porting Certificates to X.509 Structure

The certificate structure followed during implementation has almost all the fields required to form a X.509 certificate. With a few modifications it is possible to port it to standard formats. Changing it to standard structure leads to various possibilities. The whole system can be made web-enabled. Users can start using browsers to login, instead of using the GUI provided by various modules. Thus interaction of DCS with the outer world increases with all the required security features.

6.2.2 Key Migration Problem

The keys used in the current system are RSA keys for key pairs and DES keys for session keys. These keys have specific lengths associated with them. DES keys are generally 56 bit keys. As people need tighter and stronger security there may be a need for longer keys and stronger algorithms. The need for change in key lengths and algorithms result in key migration problem. The system should migrate from one key type to another in a smooth way when the changes in key structures occur. A protocol can be designed for the smooth transition of the system. The protocol could use version numbers for keys. Entities in the system can check which version of keys is in use currently and if there is a change then the entities gradually shift to the newer version.
6.2.3 Considering DCS Instance as Entities

The current system considers only users and sites as entities to which keys are assigned. These entities are authenticated mutually when they want to communicate and the communication line is made secure. In future a DCS instance itself might be considered as an entities. At present CCS provides joining and splitting conferences. So DCS instances may be assigned keys and authentication and secure communication features can be provided.

6.2.4 Re-Authentication

The current system authenticates users only during login. This can be extended to other situations where re-authentication of a user is necessary. This situation occurs when the user is about to perform a critical operation for which extreme security is required.

6.3 Summary

With this chapter, the entire requirements, design, implementation and future work for SCS module have been discussed in detail. Future workers on DCS can take hints from the above mentioned chapters, continue to work on DCS and make it a more complete system.
REFERENCES


BIOGRAPHICAL SKETCH

Ravichandran Aringunram was born in Chennai, India, in the month of July, 1979. He earned his high school diploma from Vidya Mandir senior secondary school, Chennai. He studied for his undergraduate degree in Shanmugha College of Engineering, Bharathidasan University, Trichy. He graduated in June 2000 with a Bachelor of Engineering degree with distinction. He immediately went on to pursue his master’s at the University of Florida, Gainesville, in the field of computer engineering.