SECURE SOCKET LAYER PROTOCOL SIMULATION IN JAVA

A Research Project

By

NAGENDRA KARRI

Submitted to the College of Graduate Studies
Oregon State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

JANUARY 2006

Major Subject: Computer Science

SECURE SOCKETS LAYER PROTOCOL SIMULATION IN JAVA

A Research Project

By

NAGENDRA KARRI

Approved as to style and content by:

JANUARY 2006

ABSTRACT

Author- Nagendra Karri

This project aims to study the performance of Secure Sockets Layer (SSL) Protocol implemented in JAVA for web applications. Secure Sockets Layer protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. In particular, this project focuses on an implementation of the SSL protocol used for secure data exchange between a web server (Server) and a browser (Client) through socket programming. This Secure Sockets Layer Protocol in JAVA can be executed on any machine having JAVA Virtual Machine (VM) installed.

In this project, the SSL protocol was designed to authenticate the server, and optionally the client by creating software keys on both the sides in JAVA. The authentication process uses Public-Key Encryption and Digital Signatures to verify the identity of the server. Once the server has been authenticated, the client and server use techniques of Symmetric-Key Encryption, which is very fast, to encrypt all the information they exchange for the remainder of the session and to detect any tampering that may have occurred. In this project, the maximum file size that can be encrypted and then transferred from the server to the client is 10 Megabytes.

CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The Transmission Control Protocol/Internet Protocol (TCP/IP) governs the transport and routing of data over the Internet. Other protocols such as the Hypertext Transport Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), or Internet Messaging Access Protocol (IMAP), run on top of TCP/IP in the sense that they all use TCP/IP to support typical application tasks such as displaying web pages or running email servers [1].

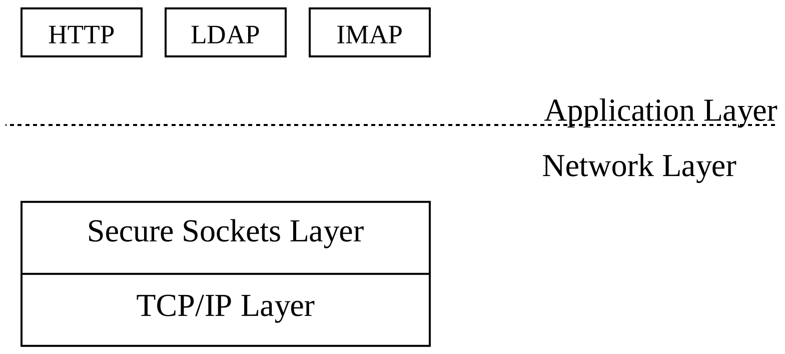


Figure 1. SSL runs above TCP/IP and below high-level application protocols

In Figure 1, the Secure Sockets Layer (SSL) protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP [1]. It uses TCP/IP on behalf of the higher-level protocols and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

The SSL protocol was originally developed by Netscape, to ensure security of data transported and routed through HTTP or LDAP application layers.

These capabilities address fundamental concerns about communication over the Internet and other TCP/IP networks:

- SSL server authentication allows a user to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a Certified Authority (CA) listed in the client's list of trusted CA's [1]. This conformation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.

- SSL client authentication allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a CA listed in the server's list of trusted CA's. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to verify the recipient's identity [1].

- An encrypted SSL connection requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering, that is, for automatically determining whether the data has been altered in transit [1].

The SSL Protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher-level protocols [2]. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent [2]. A higher-level protocol can lie on top of the SSL Protocol transparently.

1.1. Internet Security Issues

All communication over the Internet uses the TCP/IP. TCP/IP allows information to be sent from one computer to another through a variety of intermediate computers and separate networks before it reaches its destination [2].

The great flexibility of TCP/IP has led to its worldwide acceptance as the basic internet and intranet communications protocol [2]. At the same time, the fact that TCP/IP allows information to pass through intermediate computers makes it possible for a third party to interface with communications in the following ways:

- Eavesdropping: Information remains intact, but its privacy is compromised. For example, some one could learn your credit card number, or record a sensitive conversation.
- Tampering: Information in transit is changed or replaced and then sent to the recipient. For example, some one could alter an order for goods or change a person’s resume.
- Impersonation: Information passes to a person who poses as the intended recipient. Impersonation can take two forms:
    - Spoofing: A person can pretend to be some one else. For example, a person can pretend to have the email address jdoe@xyz.com or a computer can identify itself as a site called www.xyz.com when it is not. This type of impersonation is known as spoofing

Misrepresentation: A person or an organization can misrepresent itself.

For example, suppose the site www.xyz.com pretends to be a furniture store when it is really just a site that takes credit-card payments but never sends any goods.

Normally, users of the many co-operating computers that make up the Internet or other networks do not monitor or interface with the network traffic that continuously passes through their machines. However, many sensitive personal and business communications over the Internet require precautions that address the threats listed above. Fortunately, a set of well-established techniques and standards known as public-key cryptography can be employed to take such precautions [2].

Public-key cryptography facilitates the following tasks:

- Encryption and decryption allow two communicating parties to disguise information they send to each other. The sender encrypts, or scrambles, information before sending it. The receiver decrypts, or unscrambles, the information after receiving it. While in transit, the encrypted information is unintelligible to an intruder.
- Tamper detection allows the recipient of information to verify that it has not been modified in transit. Any attempt to modify data or substitute a false message for a legitimate one will be detected.
- Authentication allows the recipient of information to determine its origin, that is, to confirm the sender’s identity.
- Non-repudiation prevents the sender of information from claiming at a later date that the information was never sent [2].

SSL Basics

SSL Element

The main role of SSL is to provide security for Web traffic. Security includes confidentiality, message integrity, and authentication. SSL achieves these elements of security through the use of cryptography, digital signatures, and certificates.

Cryptography

SSL protects confidential information through the use of cryptography. Sensitive data is encrypted across public networks to achieve a level of confidentiality. There are two types of data encryption: symmetric cryptography and asymmetric cryptography (refer to Table B).

Symmetric cryptography uses the same key for encryption and decryption. An example of symmetric cryptography is a decoder ring. Alice has a ring and Bob has the same ring. Alice can encode messages to Bob using her ring as the cipher. Bob can then decode the sent message using his ring. In cryptography, the "decoder ring" is considered a pre-shared key. The key is agreed upon by both sides and can remain static. Both sides must know each other already and have agreed upon what key to use for the encryption and decryption of messages. Remember that the same key is used for encoding as well as decoding messages—thus the term *symmetric cryptography*.

Asymmetric algorithms use one key for encryption of data, and then a separate key for decryption. Asymmetric algorithms are more favorable than symmetric algorithms because even if the encryption key is learned in one direction, the third party still needs to know the other key in order to decrypt the message in the other direction.

Table B Symmetric Cryptography vs. Asymmetric Cryptography

Symmetric Cryptography
• Symmetric cryptography uses a single key for encryption and decryption.
• Symmetric cryptography requires that both parties have the key.
• Key distribution is the inherent weakness in symmetric cryptography.
• Minimal CPU cycles are required to verify keys.
• Symmetric ciphers are fortified by algorithmic strength and key lengths.
• SSL symmetric key lengths range from 40 to 168 bits.
Asymmetric Cryptography (PKI)
• Asymmetric cryptography was designed in response to the limitations of symmetric cryptography.
• Information encrypted with one key can be decrypted only with another key.
• Public key infrastructure (PKI) cryptography is up to 1000 times more CPU intensive than symmetric cryptography.
• The Rivest, Shamir, Adelman (RSA) algorithm uses modular arithmetic to enable the concept of public and private keys.
• All SSL transactions begin with an asymmetric key exchange.

With asymmetric encryption, both sides can spontaneously spawn a transaction without ever having met. This is achieved by the use of a public and private key pair. The public key of the entity is public knowledge and is used for encryption, whereas the private key of the entity remains secret and is used for decryption. PKI is the more common name for asymmetric cryptography. Although PKI is more secure, it also is more expensive in terms of processing speed. The encryption and decryption of the PKI can take up to 1000 times the processing than symmetric cryptography.

1.2. Encryption and Decryption

Encryption is the process of transforming information so it is unintelligible to anyone but the intended recipient. Decryption is the process of transforming encrypted information so that it is intelligible again [3]. A cryptographic algorithm, also called a cipher, is a mathematical function used for encryption or decryption. In most cases, two related functions are employed, one for encryption and the other for decryption.

With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a number called a key that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. Decryption with the correct key is simple. Decryption with out the correct key is very difficult, and in some cases impossible for all practical purposes.

1.2.1. Symmetric-key Encryption

With symmetric-key encryption, the encryption key can be calculated from the decryption key and vice versa [3]. With most symmetric algorithms, the same key is used for both encryption and decryption, as shown in Figure 2.

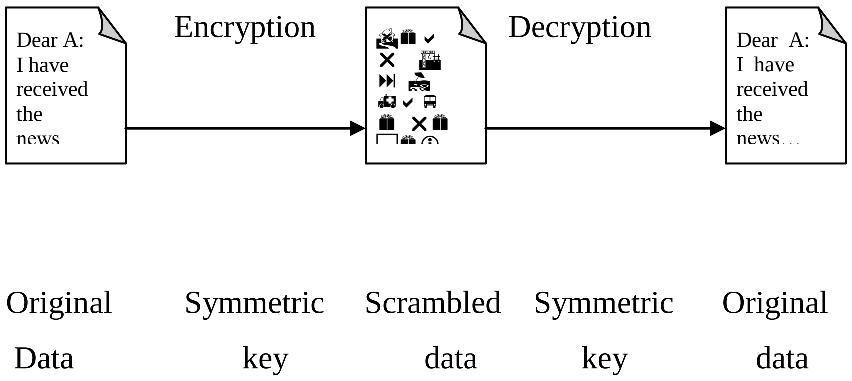


Figure 2. Symmetric-key encryption

Implementation of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption [4]. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Thus, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.

Symmetric-key encryption is effective only if the two parties involved keep the symmetric key secret. If anyone else discovers the key, it affects both confidentiality and authentication [3]. A person with an unauthorized symmetric key not only can decrypt messages sent with that key but can encrypt new messages and send them as if they came from one of two parties who were originally using the key.

Symmetric-key encryption plays an important role in the SSL protocol, which is widely used for authentication, tamper detection, and encryption over TCP/IP networks. SSL also uses techniques of public-key encryption, which is described in the next section.

1.2.2. Public-key Encryption

The most commonly used implementations of public-key encryption are based on algorithms patented by RSA Data Security [3]. Therefore, this section describes the RSA approach to public-key encryption.

Public-key encryption (also called asymmetric encryption) involves a pair of keys, a public key and a private key associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Each public key is published, and the corresponding private key is kept secret [3]. Data encrypted with your public key can be decrypted only with your private key. Figure 3 shows a simplified view of the way public-key encryption works.

This scheme lets you freely distribute a public key, and only you will be able to read data encrypted using this key. In general, to send encrypted data to someone, you encrypt the data with that person’s public key, and the person receiving the encrypted data decrypts it with the corresponding private key.

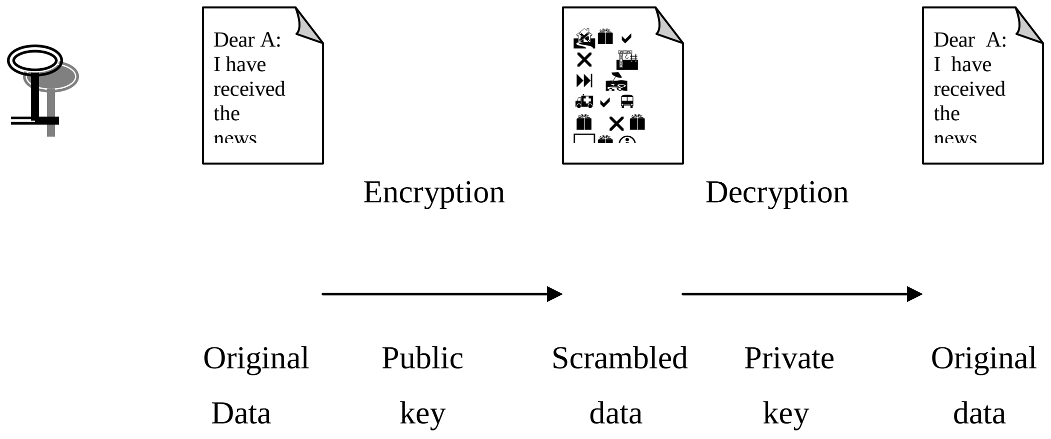


Figure 3. Public-key encryption

As it happens, the reverse of the scheme shown in Figure 3 also works: data encrypted with your private key can be decrypted only with your public key. This would not be a desirable way to encrypt sensitive data, however, because it means that any one with your public key, which by definition is published, could decrypt data. Nevertheless, private key encryption is useful, because it means you can use your private key to sign data with your digital signature, an important requirement for electronic commerce and other commercial applications of cryptography. Client software such as Communicator can then use your public key to confirm that the message was signed with your private key and that it hasn't been tampered with since it was signed [4].

1.3. Digital Signatures

To ensure message integrity, each message exchanged in SSL has a digital signature attached to it. A digital signature is a hashed message digest with public key information. The message digest is based on the checksum of the message. The message digest is difficult to reverse. Both parties compute the message digest separately and compare the hashed results. Matching results means that the checksum was unaltered during transit, minimizing the chance of a compromised message.

Encryption and decryption address the problem of eavesdropping, one of the three Internet security issues mentioned earlier. But encryption and decryption, by themselves, do not address the other two problems mentioned in Internet Security Issues: tampering and impersonation [3].

Tamper detection and related authentication techniques rely on a mathematical function called a one-way hash (also called a message digest).

A one-way hash is a number of fixed lengths with the following characteristics:

- The value of the hash is unique for the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.
- The content of the hashed data cannot, for all practical purposes, be deduced from the hash, which is why it is called “one-way.”

In public-key Encryption, it’s possible to use your private key for encryption and your public key for decryption [4]. Although this is not desirable when you are encrypting sensitive information, it is a crucial part of digitally signing any data.

Instead of encrypting the data itself, the signing software creates a one-way hash of the data, and then uses your private key to encrypt the hash. The encrypted hash, along with other information, such as the hashing algorithm, is known as a digital signature. Figure 4 shows a simplified view of the way a digital signature can be used to validate the integrity of signed data.

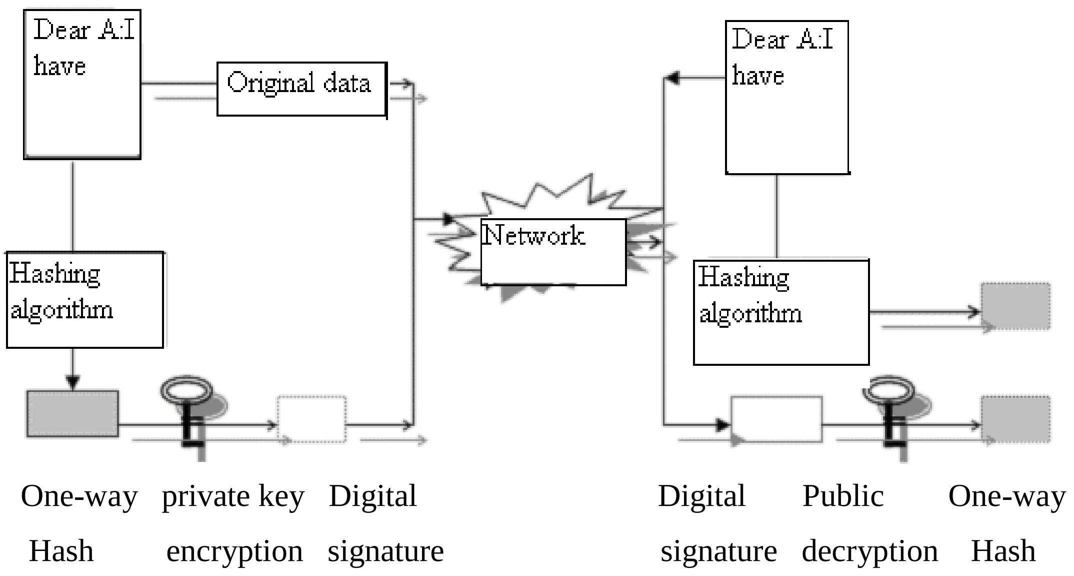


Figure 4. Using a digital signature to validate data integrity

Figure 4 shows two items transferred to the recipient of some signed data: the original data and the digital signature, which is basically a one-way hash (of the original data) that has been encrypted with the signer’s private key [4]. To validate the integrity of the data, the receiving software first uses the signer’s public key to decrypt the hash. It then uses the same hashing algorithm that generated the original hash to generate a new one-way hash of the same data.

Finally, the receiving software compares the new hash against the original hash. If the two hashes match, the data has not changed since it was signed. If they do not match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that does not correspond to the public key presented by the signer.

If the two hashes match, the recipient can be certain that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. The identity of the signer, however, also requires some way of confirming that the public key really belongs to a particular person or other entity.

1.4. Certificates and Authentication

Certificates

How do you trust the person to whom you are sending your message? SSL uses digital certificates to authenticate servers. (SSL also includes an optional authentication for clients.) Certificates are digital documents that will attest to the binding of a public key to an individual or other entity. They allow verification of the claim that a specific public key does, in fact, belong to the specified entity. Certificates help prevent someone from impersonating the server with a false key. SSL uses X.509 certificates to validate identities. X.509 certificates contain information about the entity, including public key and name. A certificate authority then validates this certificate (refer to Figure 2).

Figure 2. An X.509 Certificate

Version
Serial Number
Signature Algorithm
Issuer Name
Period of Validity • Not Before Date • Not After Date
Subject Name

Subject's Public Key • Algorithm • Public Key
Extensions
Signature

Certificate Authority

When you go to a bar or nightclub, security checks your ID to verify who you are. Your driver's license validates your ability to drive; more importantly, however, your driver's license is a trusted form of identity because a trusted third party issued your license. In the same way, a digital certificate is a mere statement of the identity of the body or individual who wishes to be authenticated. A trusted third party outside the server and client pair is needed to validate the certificate. This third party is the certificate authority. Reputable certificate authorities, such as Veri-Sign, are responsible for ensuring the trust of all World Wide Web entities.

Certificate Chaining

In some cases it may be necessary to create a chain of certificates, each one certifying the previous one until the parties involved are confident of the identity in question. This process is called certificate chaining. Certificate chaining is important in situations where the first line of certificate authorities may not be as well known or trusted as another certificate authority.

1.4.1. A Certificate identifies someone or something

A certificate is an electronic document used to identify an individual, a server, a company, or some other entity and to associate that identity with a public key. Like a driver’s license, a passport, or other commonly used personal Ids, a certificate provides generally recognized proof of a person's identity [3]. Public-key cryptography uses certificates to address the problem of impersonation.

CA's are entities that validate identities and issue certificates. They can be either independent third parties or organizations running their own certificate-issuing server software. Before issuing the certificate the, CA must use its published verification procedures for that type of certificate to ensure that an entity requesting a certificate is in fact who it claims to be. The certificate issued by the CA binds a particular public key to the name of the entity the certificate identifies (such as the name of an employee or a server). Certificates help prevent the use of fake public keys for impersonation [4]. Only the public key certified by the certificate will work with the corresponding private key possesses by the entity identified by the certificate.

In addition to a public key, a certificate always includes the name of the entity it identifies, an expiration date, the name of the CA that issued the certificate, a serial number, and other information. Most importantly, a certificate always includes the digital signature of the issuing CA. The CA's digital signature allows the certificate to function as a "letter of introduction" for users who know and trust the CA but do not know the entity identified by the certificate.

1.4.2. Authentication Confirms an Identity

Authentication is the process of confirming an identity. In the context of network interactions, authentication involves the confident identification of one party by another party. Authentication over networks can take many forms. Certificates are one way of supporting authentication.

Network interactions typically take place between a client, such as browser software running on a personal computer, and a server, such as the software and hardware used to host a Web site. Client authentication refers to the confident identification of a client by a server (that is, identification of the person assumed to be using the client software) [4]. Server authentication refers to the confident identification of a server by a client (that is, identification of the organization for a server at a particular network address).

Client and server authentication are not the only forms of authentication that certificates support. For example, the digital signature on an email message, combined with the certificate that identifies the sender, provide strong evidence that the person identified by that certificate did indeed send that message. Client authentication is an essential element of network security with in most intranets or extranets. The sections that follow contrast two forms of client authentication.

- Password-Based Authentication: Almost all server software permits client authentication by means of a name and password. For example, a server might require a user to type a name and password before granting access to the server [4]. The server maintains a list of names and passwords; if a particular name is on the list, and the user types the correct password, the server grants access.

- Certificate-Based Authentication: Client authentication based on certificates is part of the SSL protocol. The client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network [4]. The server uses techniques of public-key cryptography to validate the signature and confirm the validity of the certificate.

1.4.3. Certificate-Based Authentication

Figure 5 shows how client authentication works using certificates and the SSL protocol. When connecting to a server, a client digitally signs a randomly generated piece of data and sends both certificate and the signed data across the network. For the purposes of this discussion, the digital signature associated with some data can be thought of as evidence provided by the client to the server [5]. The server authenticates the user's identity on the strength of this evidence.

Figure 5 assumes that the user has already decided to trust the server and has requested a resource, and that the server has requested client authentication in the process of evaluating whether to grant access to the requested resource.

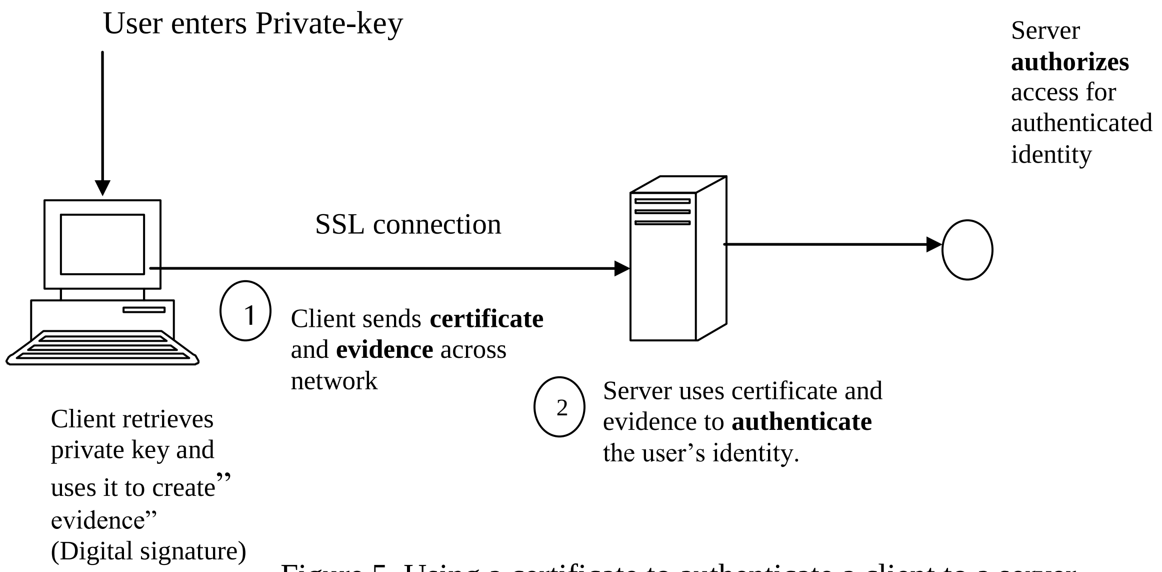

User enters Private-key
Server **authorizes** access for authenticated identity
SSL connection
1
Client sends **certificate** and **evidence** across network
2
Server uses certificate and evidence to **authenticate** the user's identity.
Client retrieves private key and uses it to create" evidence" (Digital signature)

Figure 5. Using a certificate to authenticate a client to a server

The process shown in Figure 5 requires the use of SSL. Certificate based authentication is generally considered preferable to password authentication because it is based on what the user has (the private key) as well as what the user knows (the password that protects the private key).

However, it's important to note that these two assumptions are true only if unauthorized personnel have not gained access to the user's machine or password [4]. The password for the client software is set up to request the password for the client's software private key, and the software is set up to request the password at reasonably frequent intervals.

Instead of requiring a user to send passwords across the network throughout the day, single sign-on requires the user to enter the Private Key database password just once, without sending it across the network.

For the rest of the session, the client presents the user's certificate to authenticate the user to each new server it encounters. Existing authorization mechanisms based on the authenticated user identity are not affected.

1.5 JAVA

Java was conceived by Sun Microsystems, Inc. in 1991, with the original impetus for java not being the internet, but the need for platform independent languages. Java could be used to create software that could be embedded in various consumer electronic devices. Later, java was switched from consumer electronics to internet programming; i.e the same problem that java was initially designed to solve on a small scale could also be applied to the internet on a large scale [7]. So, while the desire for an architectural-neutral programming language provided the initial spark, the internet is ultimately proved to java's large-scale success. Thus, we have chosen java as our programming language in developing the SSL Protocol, which acts between the browser and the internet. Moreover, java provides networking packages, which are used in this project .

1.5.1. Input and output in Java

The concept of streams: In Java, an object form, which we can read a sequence of bytes, is called an input stream [6]. An object to which we can write a sequence of bytes is called an output stream.

Some of the input streams supported by java are:

- Input stream,
- Buffered input stream,
- Data input stream,
- File input stream, and

- String buffer input stream.

Character input streams are virtually identical to the input streams listed above except that they operate on characters rather than on bytes. The character input stream classes are called readers instead of input streams [6]. The purpose of providing character-based versions of the input stream classes is to help facilitate the internalization of character information.

The basic reader classes provided by Java are:

- Reader,
- Buffered reader,
- File reader, and
- String reader.

The input stream class is an abstract class that serves as a base class for all other input stream classes. Input stream defines a basic interface for reading streamed bytes of information.

The buffered input stream class provides a buffered stream of input. More data is read into the buffered stream than might have been requested; subsequent reads come straight out of the buffer rather than the input device resulting in much faster read access.

The data input stream class is useful for reading primitive Java data types form an input stream in a portable fashion. The file input stream class is useful for performing simple file input.

Some of the output streams supported by Java are:

- Writer,
- Print writer,
- Buffered writer, and
- File writer.

The output stream class serves as an abstract base class for all the other output stream classes [7]. Output stream defines the basic protocol for writing streamed data to an output device.

The buffered output stream class provides a buffered stream of output. The buffered output stream class maintains a buffer that is written to when you write to stream [7]. This output approach is much more efficient because most of the data transfer takes place in memory.

The data output stream class is useful for writing primitive Java data types to an output stream in a portable fashion. The file output stream class is useful for performing simple file output.

1.5.2. Java and the net

Java supports TCP/IP both by extending the already established stream I/O interface and by adding features, required to build I/O object's across the network.

1.5.3. Socket in java

We use Universal Resource Locator (URL) and URL connection to communicate over the network at a relatively high level and for a specific purpose such as accessing resources on the internet. Sometimes our programs require lower level network communications, for example when we write a client-server application.

In client server applications, the server provides some service, such as processing database or sending out current stock prices. The client uses the service provided by the server to some end such as displaying database query results to the user or making stock purchase recommendations to an investor [7]. The communication that occurs between the client and server must be reliable and data must arrive on the client side in the same order that it was sent by the server.

TCP provides a reliable, point to point communication channel, which client server applications on the internet use to communicate. The socket and server socket classes in java.net provide a system independent communication channel using TCP.

A server application normally listens to specific port waiting for connection requests from a client [6]. When connection request arrives, the client and server establish a dedicated connection over which they can communicate.

During the connection process, the client is assigned a local port number, and binds a socket to it. The client talks to the server by writing to socket and gets information from the server by reading from it. Similarly the server gets a new local port number (it needs a new port number so that it can continue to listen for connection requests on the original port). The server also binds a socket to its local port and communicates with the client by reading from and writing to it [6]. The client and the server must agree on a protocol that is, they must agree on the language of the information transferred back and forth though the socket.
The definition of a socket is one end point of a two way communicate link between two programs running on the network [6].

The java.net package in the java development environment provides a class socket that represents one end of a two way connection between your java program and another program on the network [7]. The socket class implements the client side of two way link. If you are writing server software, you will also be interested in the server socket class which implementations the server side of the two way link.

Server socket
Java.lang.object.
Public class server socket extends object { }

The above server socket class implements server sockets. A server waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester. The actual work of the server is performed by an instance of the socket-impl class [7].

Socket

Java.lang.object

java.net.socket

public class socket extends object { }

The above client socket class implements client socket (also called just "sockets"). A socket is an endpoint for communication between two machines. The actual work of the socket is performed by an instance of the socket-impl class [7]. An application, by changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall.

CHAPTER 2
SYSTEM ARCHITECTURE

2.0. Technical Overview of the Secure Sockets Layer (SSL) Protocol

The SSL Protocol is designed to provide privacy between two communicating application's (a client and a server). Second, the protocol is designed to authenticate the server, and optionally the client [4].

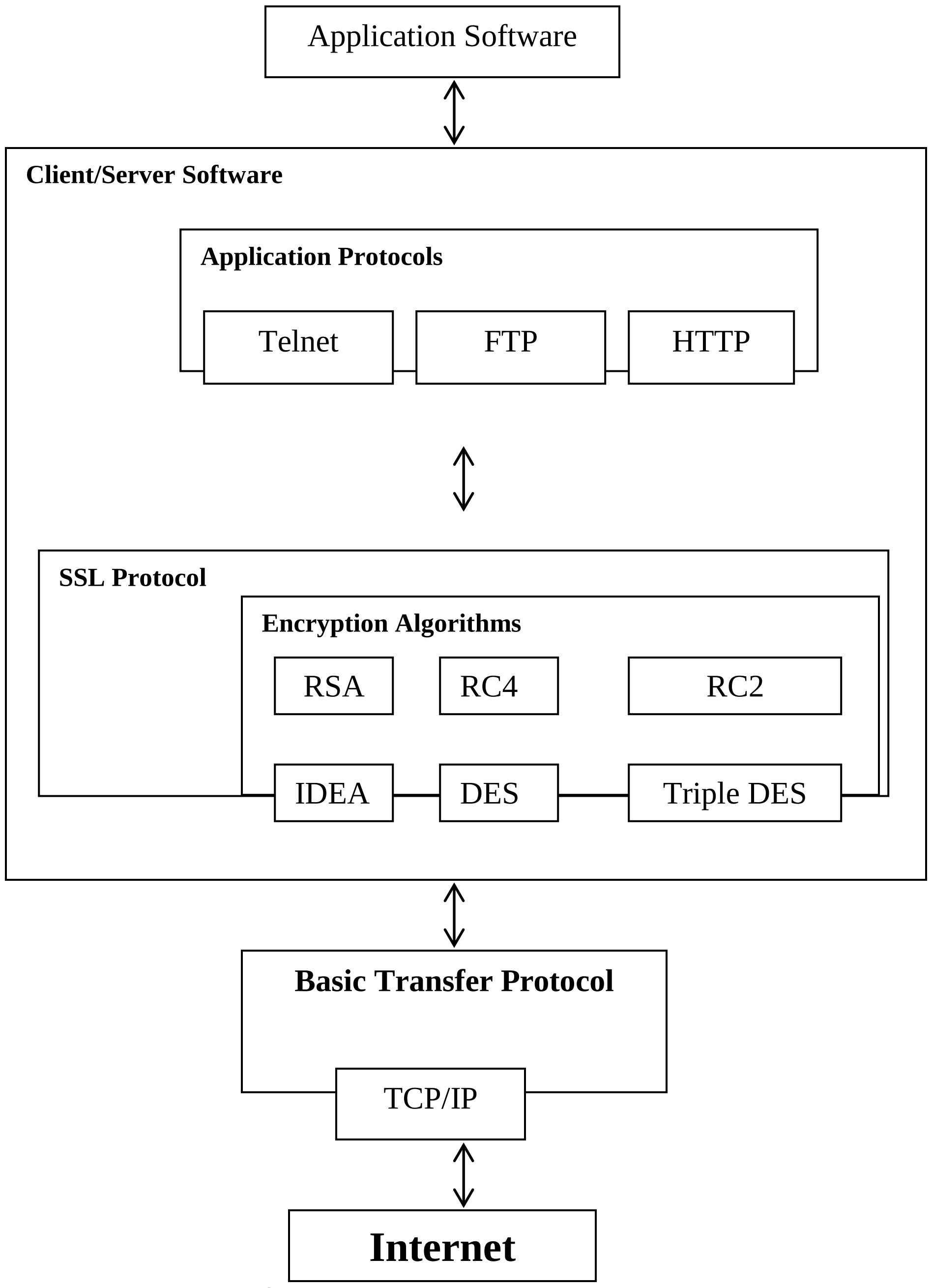


Figure 6. Diagram of relation between SSL, Applications, and the Network

SSL can layer on top of any transport protocol, it is not TCP/IP dependant, and can run under application protocols such as HTTP, FTP and TELNET and this is shown in Figure 6.

SSL uses X.509 certificates for authentication, RS as its public key cipher and one of RC4 -128, RC2-128, DES, Triple DES or IDEA as its bulk symmetric cipher [4].

2.0.1. **SSL and the ISO Reference Model**

It is important that any new communications protocol conforms to a standard model if it is to easily replace or become part of an existing protocol structure. The ISO Reference Model for Open Systems Interconnection or 7-Layer Model is the most popular abstraction [5].

In addition to information transfer, the application layer provides such services as agreement on privacy (encryption) mechanisms and authentication of an intended communication partner.

The function of the presentation layer is concerned with data security. In some applications, data sent by an application is first encrypted using a key, which is known only by the intended recipient presentation layer. The later decrypts any received data using the corresponding key before it on to the intended recipient. This is not currently part of the standard.

This compares to SSL which is conceptually split into two parts, the SSL Handshake Protocol (SSLHP) and the SSL Record Protocol (SSLRP). The SSLHP negotiates which bulk cipher is to be used and performs authentication of server and client, if requested [5]. The SSLRP packets the data into records and performs the agreed encryption on them or receives records and decrypts them.

So it can be seen that SSL actually rather neatly straddles the two ISO layers, SSLHP being at the Application Layer level and SSLRP at the Presentation Layer level.

Since security functions have not been implemented in many protocols, SSL, therefore, acts as an add-on to such protocols and not a replacement. Also, it can be seen that the use of SSL does not preclude the use of other security protocols, which operate at a higher level. For instance SHTTP (Secure Hyper Text Transfer Protocol), which is a document or data level security protocol, specifically tailored for financial transactions, may be layered over SSL.

2.1. PROBLEM DEFINITION

The primary goal of the SSL Protocol simulation in this project is to provide privacy and data integrity between two communicating applications. The simulation is composed of two layers: the SSL Record Protocol and the SSL Handshake Protocol. At the lowest level, layered on top of reliable transport protocol (e.g., TCP), is the SSL Record Protocol. In this project, the SSL Record Protocol provides connection security that has two basic properties.

- The connection is private. Symmetric cryptography is used for data encryption (e.g., DES, RC4, etc.). The keys for this symmetric encryption are generated uniquely for each connection and are based on another protocol (such as the TLS Handshake Protocol). The record Protocol can also be used without encryption.
- The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC computations. The Record Protocol can operate without a MAC, but is generally only used in this model while another protocol is using the Record Protocol as a transport for negotiating security parameters.

2.2. Hardware Requirements for the simulation of Secure Sockets Layer Protocol

- ➢ Pentium 133 MHz Processor
- ➢ 32 MB RAM (64 MB recommended)
- ➢ mouse
- ➢ IBM compatible PC
- ➢ Network: LAN

2.3. Software Requirements for the simulation of Secure Sockets Layer Protocol

- Windows NT 4.0
- JDK 1.2.1 compiler

2.4 SYSTEM DESIGN

Working of SSL:

SSL Roles

SSL has two distinct entities, server and client. The client is the entity that initiates the transaction, whereas the server is the entity that responds to the client and negotiates which cipher suites are used for encryption. In SSL, the Web browser is the client and the Web-site server is the server.

Three protocols lie within SSL, the Handshake Protocol, the Record Protocol, and the Alert Protocol. The client authenticates the server during the Handshake Protocol. When the session is initiated and the handshake is complete, the data transfer is encrypted during the Record Protocol phase. If there are any alarms at any point during the session, the alert is attached to the questionable packet and handled according to the Alert Protocol.

2.4.1. The SSL Record Protocol

SSL Records

The encryption for all messaging in SSL is handled in the Record Protocol. This protocol provides a common format to frame all Alert, ChangeCiperSpec, Handshake, and application protocol messages.

SSL records consist of the encapsulated data, digital signature, message type, version, and length. SSL records are 8 bytes long. Because the record length is fixed, encrypted messages sometimes include padding and pad length in the frame, as shown in Figure A.

Figure A. An Example of an SSL Record[4]

Type	Version	Length	
Data			
Hashed-based message authentication code (HMAC) message digest algorithm 5 (MD5)			
Pad			Pad Length

An SSL record consists of two parts, the header and the data. The header can either be 3 or 2 bytes in length, the latter being employed if there is no padding data. For a 2 byte header, the maximum record length is 32767 bytes whereas a 3 byte header will only allow a record length of up to 16383 bytes.

The data part of the record consists of a Message Authentication Code (MAC), the actual data itself and padding data, if required. It is the data part of the record which is encrypted in its entirety when encryption is necessary. Padding data is only required for use with block cipher. If a stream cipher is used or the data is already a multiple of the block size, then no padding is required and a 2 byte header record can be used. The MAC is a hash or message digest of the secret write key of the sending party, the actual data, the padding data and a sequence number in that order [3]. The sequence number is a 32-bit integer which is incremented after each message is sent.

SSL Alert Protocol

As mentioned earlier, the Alert Protocol handles any questionable packets. If either the server or client detects an error, it sends an alert containing the error. There are three types of alert messages: warning, critical, and fatal. Based on the alert message received, the session can be restricted (warning, critical) or terminated (fatal).

2.4.2. The SSL Handshake Protocol

SSL Handshake

The client always authenticates the server, and the server has the option of also authenticating the client. In general, Web servers do not authenticate the client during the Handshake Protocol because the server has other ways to verify the client other than SSL. For e-commerce, the Web-site server can verify the credit card number externally from the SSL session. In this way, the server can reserve precious processing resources for encrypted transactions.

During the Handshake Protocol, the following important steps take place: the session capabilities are negotiated, meaning the encryption (ciphers) algorithms are negotiated; and the server is authenticated to the client.

SSL uses symmetric cryptography for the bulk data encryption during the transfer phase; however, asymmetric cryptography, (that is, PKI) is used to negotiate the key used for that symmetric encryption. This exchange is critical to the Handshake Protocol. Note that the server may optionally ask the client to authenticate itself. However, it is not necessary to the protocol. Table 2 gives the steps of the Handshake Protocol.

Table 2 Handshake Protocol

1.	**Client sends *ClientHello* message.**
2.	Server acknowledges with ***ServerHello*** message
3.	Server sends its certificate
4.	Optional: Server requests client's certificate
5.	Optional: Client sends its certificate
6.	Client sends ***ClientKeyExhcange*** message
7.	Client sends ***Certificate Verify*** message

8.	Both send ***ChangeCipherSpec*** messages
9.	Both send ***Finished*** messages

The handshake protocol is composed of two phases. Phase 1 deals with the selection of a cipher, the exchange of a master key and the authentication of the server. Phase 2 handles client authentication, if requested and finishes the handshaking. After the handshake stage is complete, the data transfer between client and server begins [4]. All messages during handshaking and after are sent over the SSL Record Protocol layer.

The client_hello message sends the server some challenger-data and a list of ciphers which the client can support. The challenge-data is used to authenticate the server later on.

The server_hello message returns a connection-id, a server certificate and a modified list of ciphers which the client and server can both support. The server certificate is used by the client to obtain the servers public key and verify the identity of the server using any certification authority certificates it has [4].

The certificate message is required for any agreed-on key exchange method except anonymous Data Encryption Standard.

A server_key_exchange message may be sent if it is required. It is not required in two instances, when:

1. The server has sent a certificate with fixed DES parameters.
2. RSA key exchange is to be used.

The server_done message is sent by the server to indicate the end of the server_ hello and associated messages. After sending this message, the server will wait for a client response.

Upon receipt of the server_done message, the client should verify that the server provided a valid certificate if required and check that the server_hello parameters are valid. If all is satisfactory, the client sends one or more messages back to the server.

If the server has requested a certificate, the client begins this phase by sending a certificate message. If no suitable certificate is available, the client sends a no_certificate alert instead.

Next is the client_key_exchange message which must be sent in this phase. The content of the message depends on the type of key exchange.

Finish_phase completes the setting up of a secure connection. The client sends a change_cipher_spec message and copies the pending cipherspec into the current cipherspec. Note that this message is not considered part of the Handshake Protocol but is sent using the change_cipher_spec protocol. The client then immediately sends the finished message under the new keys [4]. The finished message verifies that the key exchange and authentication processes were successful.

2.5 SHA-1

The SHA-1 (Secure Hash Algorithm) may be used with the DSA (Digital Signature Algorithm) in electronic mail, electronic funds transfer, software distribution, data storage, and other applications which require data integrity assurance and data origin authentication [10]. SHA-1 may also be used whenever it is necessary to generate a condensed version of a message. The algorithm for SHA-1 which is used in this project is explained in detail below.

A *hash function H* is a transformation that takes an input m and returns a fixed-size string, which is called the hash value *h* (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually chosen to have some additional properties.

The basic requirements for a cryptographic hash function are as follows.

- The input can be of any length.
- The output has a fixed length.
- $H(x)$ is relatively easy to compute for any given *x*.
- $H(x)$ is one-way.
- $H(x)$ is collision-free.

A hash function *H* is said to be *one-way* if it is hard to invert, where ``hard to invert'' means that given a hash value *h*, it is computationally infeasible to find some input *x* such that $H(x) = h$. If, given a message *x*, it is computationally infeasible to find a message *y* not equal to *x* such that $H(x) = H(y)$, then *H* is said to be a *weakly collision-free* hash function. A *strongly collision-free* hash function *H* is one for which it is computationally infeasible to find any two messages *x* and *y* such that $H(x) = H(y)$.

Explanation: This Standard specifies a Secure Hash Algorithm, SHA-1, for computing a condensed representation of a message or a data file. When a message of any length $< 2^{64}$ bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then be input to the Digital Signature Algorithm (DSA) which generates or verifies the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature.

The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify. SHA-1 is a technical revision of SHA (FIPS 180). The SHA-1 is based on principles similar to those used by Professor Ronald L. Rivest of MIT when designing the MD4 message digest algorithm ("The MD4 Message Digest Algorithm," and is closely modeled after that algorithm.

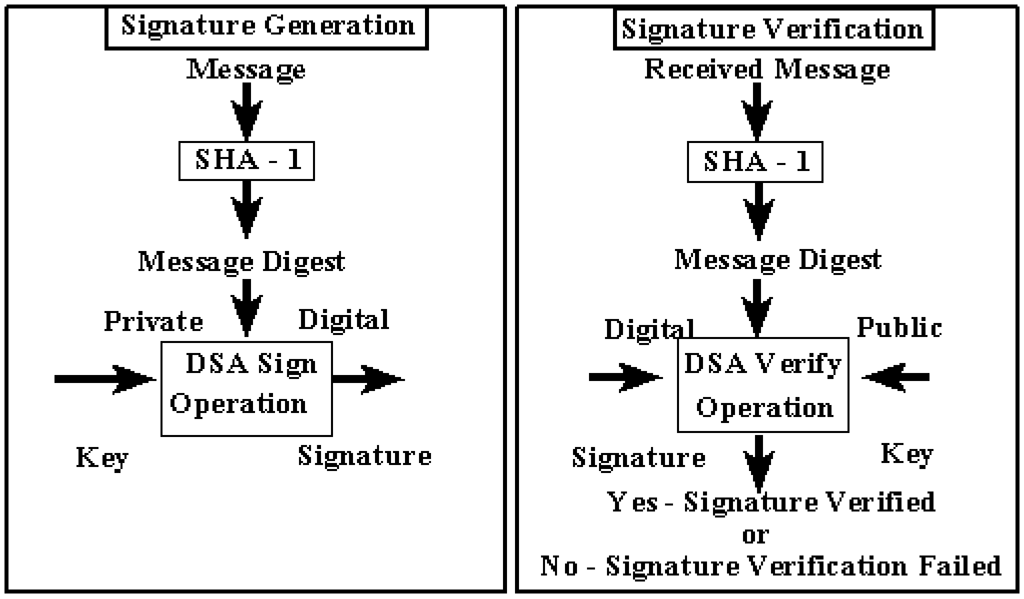


Figure 10: Using the SHA-1 with the DSA

Implementations: The SHA-1 may be implemented in software, firmware, hardware, or any combination thereof. Only implementations of the SHA-1 that are validated by NIST will be considered as complying with this standard. Information about the requirements for validating implementations of this standard can be obtained from the National Institute of Standards and Technology, Computer Systems Laboratory.

Algorithm for SHA-1

1) The message is divided into 64 byte blocks.

- Message is less than 2^{61} bytes.
- Message will be padded to be a multiple 64 bytes.

2) A key of 5 constants is chosen H_0-H_5. Each constant is 4 bytes long.

3) Divide block M (i) into 16 words with W (0), W (1), ………, W (15), where W (0) is the left-most word.

4) Create words 16 to 79 with

W(t)= S^1 (W(t-3) XOR W (t-8) XOR W (t-14) XOR W (t-16)).

Here a block contains only 16 words. Hence this operation extends over multiple blocks.

```
5) Let A=$H_0$, B=$H_1$, C=$H_2$, D=$H_3$, E=$H_4$
6) For t=0 to 79 do:
        {
            TEMP=$S^5$ (A) +$f_1$ (A, B, C, D) +E+W (t) +K (t);
            E=D; D=C; C=$S^{30}$ (B); B=A; A=TEMP;
        }
7) Let $H_0$=$H_0$+A; $H_1$=$H_1$+B; $H_2$=$H_2$+C; $H_3$=$H_3$+D; $H_4$=$H_4$+E;
8) Repeat steps 1 to 7 on next 64-byte block.
```

After processing M (n), the message digest is the 160-bit string represented by the 5 words

H_0 H_1 H_2 H_3 H_4.

Applications: The SHA-1 may be used with the DSA in electronic mail, electronic funds transfer, software distribution, data storage, and other applications which require data integrity assurance and data origin authentication. The SHA-1 may also be used whenever it is necessary to generate a condensed version of a message.

2.6. RSA

[Rivest, Shamir and Adleman 1978] invented an asymmetric cryptosystem named MIT cryptosystem. However, the name RSA is usually used today instead of MIT cryptosystem. In contrast to DES, RSA's security relies on a solid mathematical background. Although current RSA implementations are much slower than symmetric cryptosystems, RSA is not only used for key exchange. Increasing performance of computers could make RSA to a real alternative to symmetric cryptosystems for numerous applications.

An "RSA operation," whether encrypting, decrypting, signing, or verifying is essentially a modular exponentiation. This computation is performed by a series of modular multiplications.

In practical applications, it is common to choose a small public exponent for the public key. In fact, entire groups of users can use the same public exponent, each with a different modulus. (There are some restrictions on the prime factors of the modulus when the public exponent is fixed.) This makes encryption faster than decryption and verification faster than signing. With the typical modular exponentiation algorithms used to implement the RSA algorithm, public key operations take $O(k^2)$ steps, private key operations take $O(k^3)$ steps, and key generation takes $O(k^4)$ steps, where k is the number of bits in the modulus. ``Fast multiplication" techniques, such as methods based on the Fast Fourier Transform (FFT), require asymptotically fewer steps. In practice, however, they are not as common due to their greater software complexity and the fact that they may actually be slower for typical key sizes.

A Simple explanation of RSA Algorithm in view to computer:

Let p and q be distinct large primes and let n be their product. Assume that we also computed two integers, d (for decryption) and e (for encryption) such that d * e 1 (mod ø (n)) where *ø (n)* is the number of positive integers smaller than n that have no factor except 1 in common with *n.*

The integers *n* and *e* are made public, while *p*, *q*, and *d* are kept secret. Let *m* be the message to be sent, where m is a positive integer less than and relatively prime to *n*. A plaintext message is easily converted to a number by using either the alphabet position of each letter (a=01, b=02, ..., z=26) or using the standard ASCII table. If necessary (so that *m*<*n*), the message can be broken into several blocks. The encoder computes and sends the number m' = *m^e* mod *n.* To decode, we simply compute *e^d* mod *n.* The security of RSA depends on the fact that it takes an impractical amount of time to factor large numbers

This is by no means a comprehensive explanation of how RSA works, nor is it meant to be. The security of RSA is based on the difficulty of factoring large numbers, which is next to impossible for 1,024-bit numbers today. This could change tomorrow, however, as technology develops.

In the RSA cryptosystem, the participants create their public and secret keys with the following procedure:

ALGORITHM FOR RSA

1. Select randomly two large prime numbers p and q.
2. Compute n by the equation n= pq.
3. Select a small odd integer e that is relatively prime to (p-1) (q-1).
4. Computed d as the multiplicative inverse of e, modulo (p-1) (q-1).
5. Publish the pair P= (e; n) as the RSA public key.
6. Keep the secret pair S= (d; n) as the RSA secret key.

The transformation of a message M associated with the public key pair P= (e; n) is

$$P\ (M) = M_e (\text{mod } n).$$

The transformation of a cipher text C associated with a secret key pair S= (d; n) is

$$S(C) = C_d (\text{mod } n).$$

In practice, the RSA system is often used together with a secret-key cryptosystem, such as DES, to encrypt a message by means of an RSA digital envelope

2.7. DES

DES, an acronym for the Data Encryption Standard, is the name of the Federal Information Processing Standard (FIPS) 46-3, which describes the data encryption algorithm (DEA). The DEA is also defined in the ANSI standard X3.92.

DEA is an improvement of the algorithm Lucifer developed by IBM in the early 1970s. While IBM essentially designed the algorithm, the NSA and NBS (now NIST) played a substantial role in the final stages of the development. The DEA, often called DES, has been extensively studied since its publication and is the best known and widely used symmetric algorithm in the world.
The DEA has a 64-bit block size and uses a 56-bit key during execution (8 parity bits are stripped off from the full 64-bit key). The DEA is a symmetric cryptosystem, specifically a 16-round Feistel cipher and was originally designed for implementation in hardware. When used for communication, both sender and receiver must know the same secret key, which can be used to encrypt and decrypt the message, or to generate and verify a message authentication code (MAC). The DEA can also be used for single-user encryption, such as to store files on a hard disk in encrypted form. In a multi-user environment, secure key distribution may be difficult; public-key cryptography provides an ideal solution to this problem.

NIST has re-certified DES (FIPS 46-1, 46-2, 46-3) every five years. FIPS 46-3 reaffirms DES usage as of October 1999, but single DES is permitted only for legacy systems. FIPS 46-3 includes a definition of triple-DES (TDEA, corresponding to X9.52); TDEA is "the FIPS approved symmetric algorithm of choice." Within a few years, DES and triple-DES will be replaced with the Advanced Encryption Standard (AES).

Speed of DES

By comparison, DES and other block ciphers are much faster than the RSA algorithm. DES is generally at least 100 times as fast in software and between 1,000 and 10,000 times as fast in hardware, depending on the implementation. Implementations of the RSA algorithm will probably narrow the gap a bit in coming years, due to high demand, but block ciphers will get faster as well.

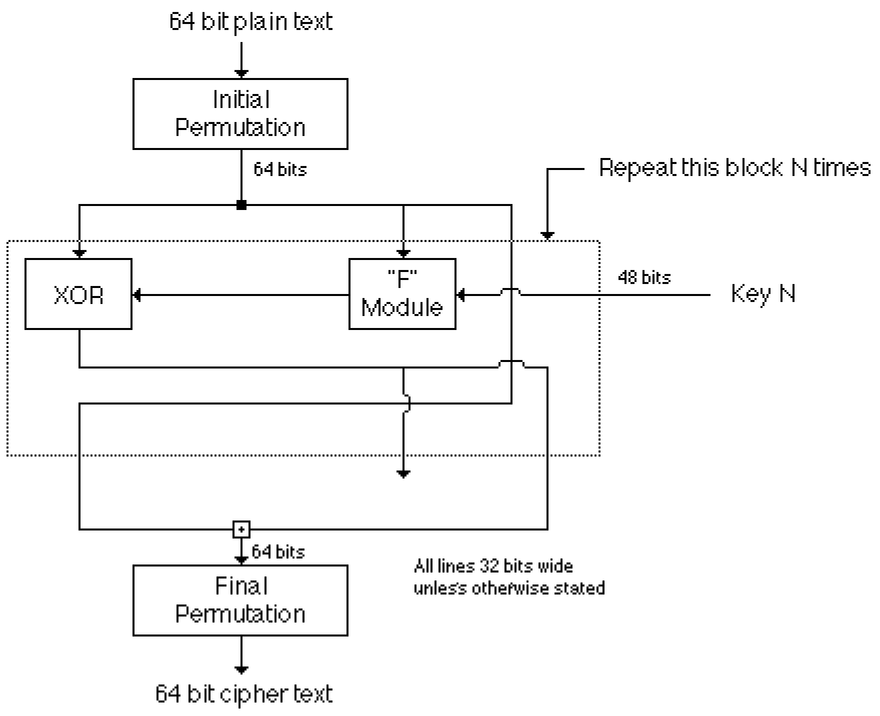
Usage of DES

When using DES, there are several practical considerations that can affect the security of the encrypted data. One should change DES keys frequently, in order to prevent attacks that require sustained data analysis. In a communications context, one must also find a secure way of communicating the DES key from the sender to the receiver. Use of the RSA algorithm or some other public-key technique for key management solves both these issues: a different DES key is generated for each session, and secure key management is provided by encrypting the DES key with the receiver's public key. The RSA system, in this circumstance, can be regarded as a tool for improving the security of DES (or any other secret-key cipher).
If one wishes to use DES to encrypt files stored on a hard disk, it is not feasible to frequently change the DES keys, as this would entail decrypting and then re-encrypting all files upon each key change. Instead, one might employ a master DES key that encrypts the list of DES keys used to encrypt the files; one can then change the master key frequently without much effort. Since the master key provides a more attractive point of attack than the individual DES keys used on a per file basis, it might be prudent to use triple-DES as the encryption mechanism for protecting the file encryption keys.

DES can be used for encryption in several officially defined modes, and these modes have a variety of properties. ECB (electronic codebook) mode simply encrypts each 64-bit block of plaintext one after another under the same 56-bit DES key. In CBC (cipher block chaining) mode, each 64-bit plaintext block is bitwise XORed with the previous cipher text block before being encrypted with the DES key. Thus, the encryption of each block depends on previous blocks and the same 64-bit plaintext block can encrypt to different cipher text blocks depending on its context in the overall message. CBC mode helps protect against certain attacks, but not against exhaustive search or differential cryptanalysis. CFB (cipher feedback) mode allows one to use DES with block lengths less than 64 bits.

In practice, CBC is the most widely used mode of DES, and it is specified in several standards. For additional security, one could use triple encryption with CBC.

DES Block Diagram



ALGORITHM FOR DES

1. Process the key

1.1. Get a 64-bit key from the user. (Every 8th bit is considered a parity bit. For a key to have correct parity, each byte should contain an odd number of "1" bits.)

1.2. Calculate the key schedule.

1.2.1. Perform the following permutation on the 64-bit key by using the following table.

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

The parity bits are discarded, reducing the key to 56 bits. Bit 1 of the permuted block is bit 56 of the original key, bit 2 is bit 49, and so on with bit 56 being bit 4 of the original key. Split the permuted key into two halves. The first 28 bits are called C [0] and the last 28 bits are called D [0].

1.2.2 Calculate the 16 sub keys. Start with i=1.

1.2.2.1 Perform one or two circular left shifts on both C [i-1] and D [i-1] to get C[i] and D[i], respectively.

1.2.2.2 The numbers of shifts per iteration are calculated.

1.2.2.3 Permute the concatenation C[i] D[i] as indicated below. This will yield K[i], which is 56 bits long.

1.2.2.4 Loop back to 1.2.3.1. Until K [16] has been calculated.

2. Process a 64-bit data block

2.1 Get a 64-bit data block. If the block is shorter than 64 bits, it should be padded as appropriate for the application.

2.2 Perform the following permutation on the data block.

2.3 Split the block into two halves. The first 32 bits are called L [0], and the last 32 bits are called R [0].

2.4 Apply the 16 sub keys to the data block. Start with i=1.

2.4.1 Expand the 32-bit R [i-1] into 48 bits.

This is done by using a selection table that repeats some of the bits in $\boldsymbol{R_{n-1}}$. We'll call the use of this selection table the function **E**. Thus **E**($\boldsymbol{R_{n-1}}$) has a 32 bit input block, and a 48 bit output block.

Let **E** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E BIT-SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus the first three bits of **E**($\boldsymbol{R_{n-1}}$) are the bits in positions 32, 1 and 2 of $\boldsymbol{R_{n-1}}$ while the last 2 bits of **E**($\boldsymbol{R_{n-1}}$) are the bits in positions 32 and 1.

2.4.2 Exclusive-or E(R [i-1]) with K[i].

2.4.3 Break E(R [i-1]) xor K[i] into eight 6-bit blocks. Bits -6 are B [1], bits 7-12 are B [2], and so on with bits 43-48 being B [8].

2.4.4 Substitute the values found in the S-boxes for all B[j]. Start with j=1. All values in the S-boxes should be considered 4 bits wide.

2.4.4.1 Take the 1st and 6th bits of B[j] together as a 2-bit value (call it m) indicating the row in S[j] to look in for the substitution.

2.4.4.2 Take the 2nd through 5th bits of B[j] together as a 4-bit value (call it n) indicating the column in S[j] to find the substitution.

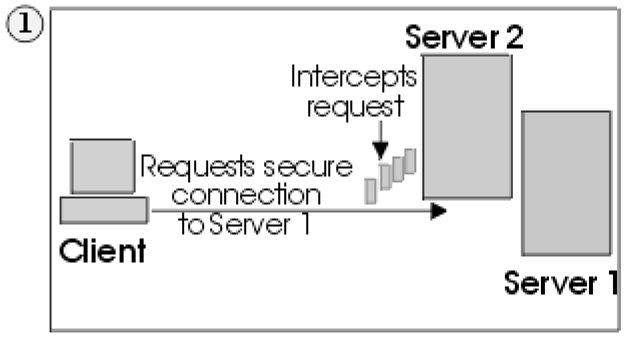
2.4.4.3 Replace B[j] with S[j][m][n].

Chapter-3
SSL and Security

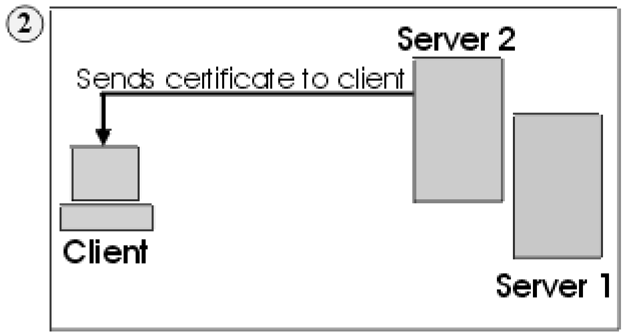
Secure Sockets Layer (SSL) protocol: The SSL creates a standard SSL connection between the client and the server. The client contacts the server and checks to make sure that the server has a valid certificate. This type of connection ensures that all data exchanged between client and server is encrypted, and is therefore not readable by a third party on the Internet.

3.1. Server Authentication (SSL)

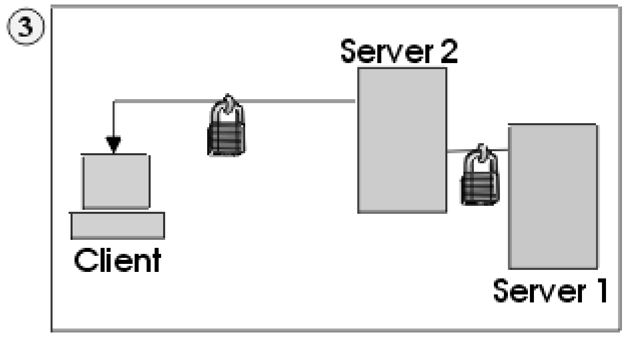
SSL by itself does not guarantee that the client is communicating with the correct server. To illustrate the risks involved with this protocol, consider the following scenario. There are two servers, Server1 (hod.S1.com) and Server2 (hod.S2.com), and one client, Client. Both servers have valid certificates from a CA that the client trusts. Client wants a secure session with Server1, but Server2 wants to eavesdrop on their communication, and is physically located in such a place that it can do so. The scenario goes as follows:



Client sends a request for an SSL session to Server1. The request (and all subsequent traffic) actually goes through Server2. Instead of forwarding Client's request to Server1, Server2 responds directly to the request by sending its own certificate to Client.



Client receives Server2's certificate and checks its list of trusted CAs. Since Server2's certificate is signed by the same CA as Server1's certificate, Client accepts the certificate and creates a secure session with Server2.



Having completed the secure session with Client, Server2 requests and creates its own SSL session with Server1. From this point, Client sends encrypted information to Server2. Server2 decrypts the information, re-encrypts it, and then sends it to Server1. It does the same for information flowing in the opposite direction. The result is that, although all data is encrypted when it flows over the Internet, Server2 is able to read it, and even change it.

To help avoid this danger, the Server Authentication (SSL) option is provided. When this is used, the client, after making sure that the server's certificate can be trusted, checks whether the Internet name in the certificate matches the Internet name of the server. If they match, the SSL negotiation will continue. If not, the connection ends immediately.

For this check to be valid and give a positive result, two conditions must be met:

- The client must be locally-installed. A client downloaded using http cannot be trusted for server authentication. If server authentication is of vital importance, you should use only locally-installed clients or use https on your Web server.
- The common name in the server's certificate must match its Internet name.

3.2. Elements that work together to establish a secure SSL connection

Client: The client needs to be an FTP client with SSL capabilities.

Certificate: Certificates are digital identification documents that allow both servers and clients to authenticate each other. A certificate file has a .crt extension. Server certificates contain information about your company and the organization that issued the certificate (such as Verisign or Thawte) while client certificates contain information about the user and the organization that signed the certificate. You can choose to either trust or distrust a certificate. In some cases, the client's certificate must be signed by the server's certificate in order to open an SSL connection.

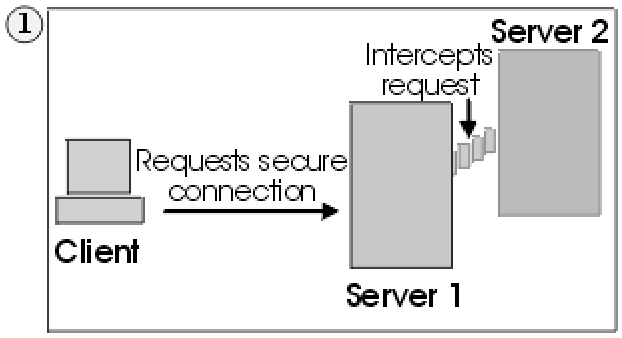
Session Key: The client and the server use the session key to encrypt data. It is created by the client via the server’s public key.

Public Key: The client encrypts a session key with the server’s public key. It does not exist as a file, but is produced when a certificate and private key are created.

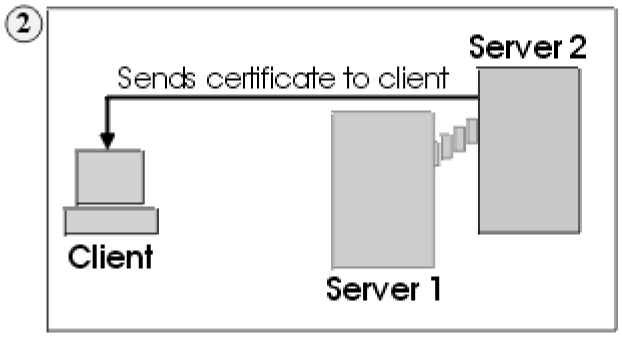
Private Key: The server's private key decrypts the client's session. The private key has a .key extension and is part of the public-private key pair.

Certificate Signing Request: A certificate signing request is generated each time a certificate is created. A certificate signing request has a .csr extension. This file is used when you need to have your certificate signed. Once the Certificate Signing Request file is signed, a new certificate is made and can be used to replace the unsigned certificate.

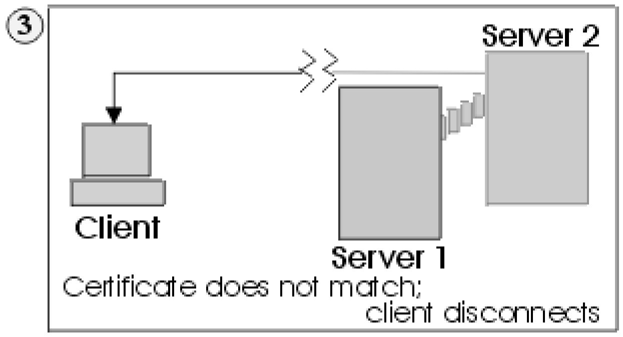
With SSL enabled, the security scenario would proceed as follows:



1. Client sends a request for an SSL session to Server1. The request (and all subsequent traffic) actually goes through Server2. Instead of forwarding Client's request to Server1, Server2 responds directly to Client's request by sending its own certificate to Client.



2. Client receives Server2's certificate and checks its list of trusted CAs. Since Server2's certificate is signed by the same CA as Server1's certificate, Client accepts the certificate and creates a secure session with Server2.



3. After the secure session has been completed, but before any real data has been sent or received, Client compares the Internet name in the certificate it received (hod.S2.com) with the name of the server it wants to talk to (hod.S1.com). Since they do not match, Client knows that the connection should not continue and disconnects it.

CHAPTER 4
SOFTWARE MODULES AND IMPLEMENTATION

In this project, I wrote the server.java, client.java, rsa.java, des.java, and sha1.java programs. In this project, the server.java program is initially run from the command prompt. This program will be waiting for any request from the client. Then, the files rsa.java, dsa.java and sha1.java programs are compiled together. At which time, the main program, client.java is compiled and run from another command prompt. This program contains the code for the user interface. In the client.java program, any existing text file can be uploaded and can then be encrypted using the des.java program. The encryption of the text file is done with the help of hash functions in the sha1.java program. After the encryption of the text file, the encrypted file is sent to the destination server with the help of the rsa.java program which sees that the encrypted file is sent only to the destination server and not to any other server by locking the target server's Internet Protocol address. In this project any text file up to a size of 10MB can be uploaded and encrypted and transmitted safely. Then after the transmission of the file to the target server, the encrypted file is opened on the target machine and is decrypted using the same hash functions found in sha1.java program. After decryption, the original data in the text file can be viewed on the destination server.

When retrieving a single file from the server, different protocols have different performance issues. The bandwidth efficiency varies considerably for each protocol as the file size is increased from 10Bytes to 10 Mega-Bytes. The below figure shows the performance of HTTPS (HTTP over SSL) compared to different protocols when used in the same context. Here we observe that as the file size increases, the bandwidth efficiency of HTTP when used with SSL is far better compared to the bandwidth efficiency of HTTP when used alone in the same context. Also the bandwidth efficiency of HTTPS dips after reaching a threshold value whereas in the case of HTTP, it reaches a constant value after some time and remains the same throughout.

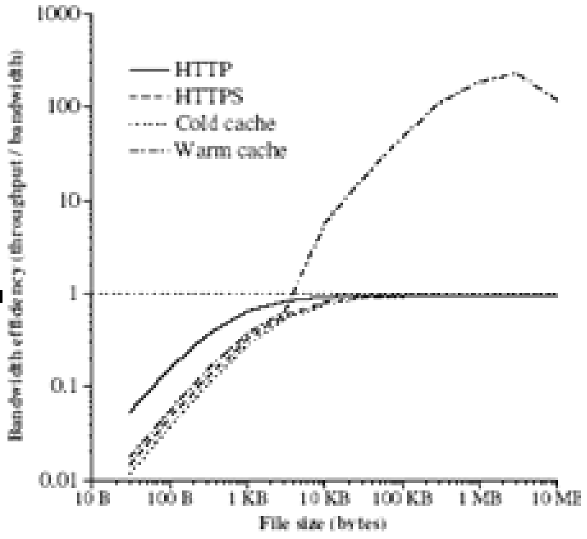


Figure 7: Ratio of throughput achieved to the bandwidth used when retrieving a single file from the server.

Figure 8 shows this data as the "savings factor", the ratio of bandwidth consumed by SSL over HTTP and HTTPS

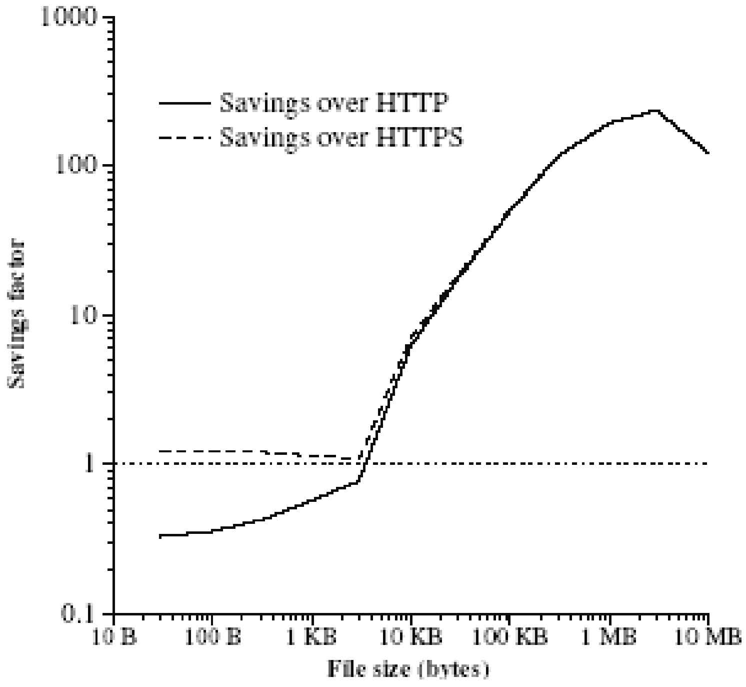


Figure 8: Bandwidth Savings of SSL Splitting over HTTP and HTTPS

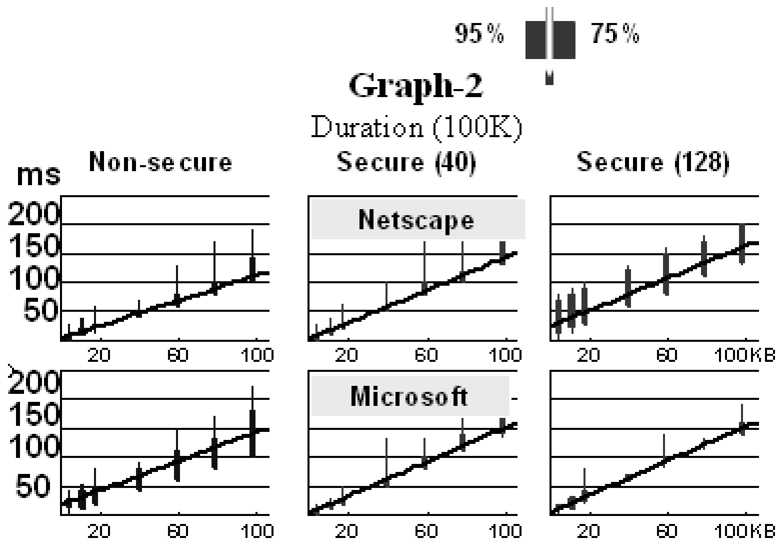


Figure 9: Comparison of HTTP and HTTPS over 100K file size.

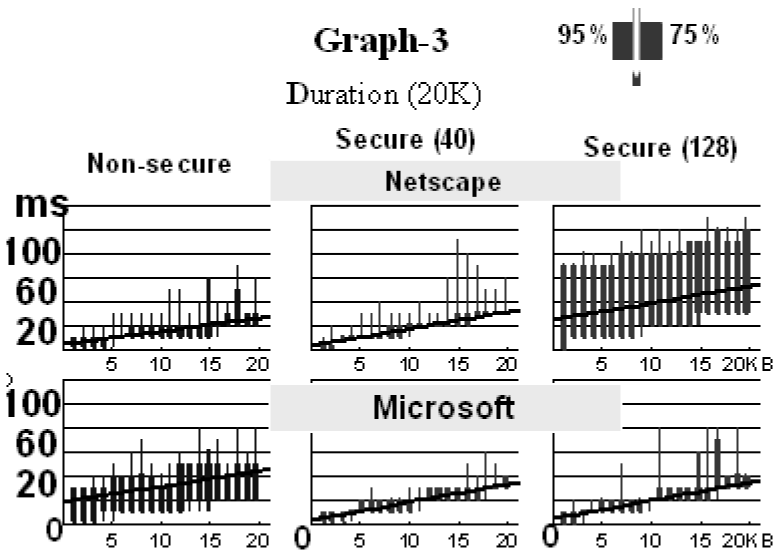


Figure 10: Comparison of HTTP and HTTPS over 20K file size.

4.1. List of java files which I developed and their uses in coding:

- SERVER.JAVA is the program which will be always running for any requests from the Clients. The Java code for the server program is in Appendix A.

- CLIENT.JAVA is the main program. This contains the code for the user interface. The Java code for the client program is in Appendix B.

- SHA1.JAVA contains the code for the SHA1 (Secure Hash) Algorithm cryptographic Hash Generation [10]. The Java code for the SHA1 algorithm is in Appendix C.

- RSA.JAVA contains the code for RSA (Raviest Shamir Adleman) Public Key Algorithm which is used for Authentication [9]. The Java code for the RSA algorithm is in Appendix D.

- DES.JAVA contains the code for Data Encryption Standard (DES) Algorithm which is a symmetric key algorithm [12]. The Java code for the DES algorithm is in Appendix E.

4.2 Testing

In the first phase, I opened the command prompt and compiled the server.java program using the command, javac server.java. Then after compilation, I executed the server class file using the command, java server.

In the second phase, I simultaneously opened another command prompt and compiled rsa.java, sha1.java, des.java and client.java programs in sequence using the commands, javac rsa.java, javac sha1.java, javac des.java, javac client.java. Then after compilation of all the above mentioned files, I executed the client class file using the command, java client.

In the third phase, I opened the notepad terminal and then created a text file of size 10MB. Then from the running client program, I encrypted this text file with the help of des.java and sha1.java which contains the encryption techniques. The encryption techniques consist of generating hash functions. In this project, I used the numbers 15, 7 and 1 in generating the encrypted format. These numbers are fed from the command prompt to the client program for generating encrypted text.

In the fourth phase, I sent the encrypted file to the server program that is running in the command prompt and then from the server program, I opened the encrypted file and then decrypted using the same hash functions used for encryption and the obtained the original text file.

One can observe the latency for different file sizes in order to retrieve it from the server by using different protocols. HTTPS (HTTP over SSL) and HTTP have similar latencies after the file size increases beyond 100KB.

The graph of latencies versus file size is shown in Figure 12. It shows three clear lines, one for each of HTTP, HTTPS, and SSL

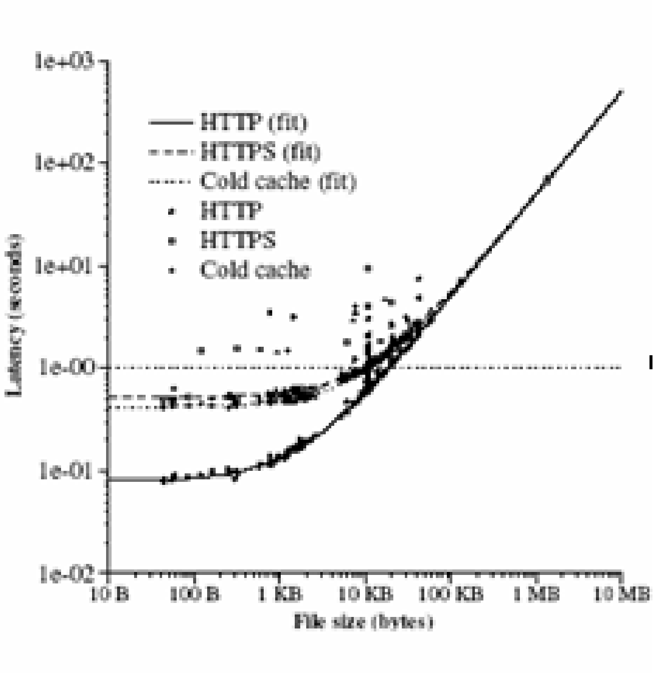


Figure 11: Distribution of Latencies versus File Sizes.

The main advantage of these software modules, which I developed for encryption, is that, I was able to encrypt a file of size 10MB at a time without the need of splitting the file into smaller sizes and then encrypt them. By using these software modules, which I developed, one can overcome the need of splitting files of sizes 10MB into smaller files and then encrypt. One can directly run the above programs and encrypt and send files of 10MB at a time thus eliminating the need of splitting the file into smaller sizes and then encrypting each piece of the original file. Also there is one more advantage in using the software modules developed by me. While decrypting the encrypted file on the destination computer, the whole file can be decrypted at once without the need for waiting for the splitted encrypted files which are still to be sent by the client. The client still has to wait for the complete transmission of data packets of the complete file from the server before decryption can start.

Chapter-5
CONCLUSION

This project was implemented using the Secured Sockets Layer Protocol in JAVA. In this project, I created an encrypted link and demonstrated a file transfer through this link between the server and a client using Data Encryption Standard and Raviest Shamir Adleman Encryption Standards.

In this project, the SSL protocol was designed to authenticate the server, and optionally the client by creating software keys on both the sides in JAVA. The authentication process uses Public-Key Encryption and Digital Signatures, which are stored in buffers to confirm that the server in fact is the server it claims to be. Once the server has been authenticated, the client and server use techniques of Symmetric-Key Encryption, which is very fast, to encrypt all the information they exchange for the remainder of the session and to detect any tampering that may have occurred. In this project, the maximum file size that can be encrypted and then transferred from the server to the client is 10 Megabytes.

REFERENCES

1. http://www.windowsecurity.com/articles/Secure_Socket_Layer.html
2. http://www.eventhelix.com/RealtimeMantra/Networking/SSL.pdf
3. http://wp.netscape.com/eng/ssl3/3-SPEC.HTM#7-1
4. http://pirate.shu.edu/~jenninju/InternetLaw/10_UCITA/SSL.pdf
5. Andrew S.Tanenbaum,"Computer Networks", Fourth Edition, Prentice Hall PTR.
6. Herbert Schildt, "The Complete Reference: JAVA2 ", 3rd Edition, published by Tata McGraw Hill.
7. Elliotte Rusty Harold, "Java Network Programming", 3rd Edition.
8. William Stallings, "Network Security Essentials: Applications and Standards", Second Edition.
9. R. Rivest, A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems ", Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126.
10. NIST FIPS PUB 180-1, "Secure Hash Standard ", National Institute of Standards and Technology, U.S. Department of Commerce, DRAFT, 31 May 1994.
11. W. Tuchman, "Hellman Presents No Shortcut Solutions to DES ", IEEE Spectrum, v. 16, n. 7, July 1979, pp40-41.
12. ANSI X3.106, "American National Standard for Information Systems-Data Link Encryption ", American National Standards Institute, 1983.

13. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001 Section 31.7: The RSA public-key cryptosystem, pp.881–887.
14. Eli Biham, Rafi Chen, Near-Collisions of SHA-0, Cryptology ePrint Archive, Report 2004/146, 2004 (to appear CRYPTO 2004).
15. Florent Chabaud, Antoine Joux: Differential Collisions in SHA-0. CRYPTO 1998. pp56–71.
16. Henri Gilbert, Helena Handschuh: Security Analysis of SHA-256 and Sisters. Selected Areas in Cryptography 2003: pp175–193.
17. http://www.tropsoft.com/strongenc/des.htm
18. Xiaoyun Wang, Hongbo Yu and Yiqun Lisa Yin, Efficient Collision Search Attacks on SHA-0, CRYPTO 2005.
19. Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu, Finding Collisions in the Full SHA-1, CRYPTO 2005.
20. http://www.eventid.net/docs/desexample.htm
21. Data Encryption Standard, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. (January 1977).
22. Carl H. Meyer and Stephen M. Matyas, Cryptography: A New Dimension in Computer Data Security, John Wiley & Sons, New York, 1982.
23. Dorthy Elizabeth Robling Denning, Cryptography and Data Security, Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

24. D.W. Davies and W.L. Price, Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronics Funds Transfer, Second Edition, John Wiley & Sons, New York, 1984, 1989.
25. Miles E. Smid and Dennis K. Branstad, "The Data Encryption Standard: Past and Future," in Gustavus J. Simmons, ed., Contemporary Cryptography: The Science of Information Integrity, IEEE Press, 1992.
26. http://tn3270.umsystem.edu/en/help/serverauth.html
27. Douglas R. Stinson, Cryptography: Theory and Practice, CRC Press, Boca Raton, 1995.
28. http://www.schneier.com/blog/archives/2005/08/new_cryptanalyt.html
29. http://www.sei.cmu.edu/str/descriptions/clientserver_body.html
30. http://www.cisco.com/en/US/netsol/ns340/ns394/ns50/ns140/networking_solutions_white_paper09186a0080136858.shtml
31. Bruce Schneider, Applied Cryptography, Second Edition, John Wiley & Sons, New York, 1996.
32. http://www.rsasecurity.com/rsalabs/node.asp?id=2176
33. http://www.iccd-conference.org/proceedings/2000/08010007.pdf
34. http://www.thenextwave.com/page19.html
35. http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/csec_ssl.html
36. http://help.globalscape.com/help/secureserver2/About_SSL.htm
37. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, 1997.

38. http://www.rsasecurity.com/rsalabs/node.asp?id=2228

39. http://www.iaik.tu-graz.ac.at/research/publications/theses/schindler.pdf

40. http://www.linuxjournal.com/article/6695