Securing Web Communications

A Thesis
in TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Engineering

by

Lim Vu

April 25, 2002

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses.

____

Approved ______________________________________ (Technical Advisor)
David Evans

Approved ______________________________________ (TCC Advisor)
Betsy Mendelsohn

Preface

Succinctly stated, this thesis works towards securing regular web communications such as instant messaging and file sharing. One might ask, is this an exercise in futility? Who is going to spy on instant messaging conversations? Perhaps no one has or ever will. However, this thesis does not strive simply to secure messaging. It presents serious issues concerning information security and privacy in our digitally connected world. Thus, this thesis attempts an exercise in securing web communications so that those reading this paper and I, the author, can learn greater lessons about implementing secure, efficient security solutions for the burgeoning information and privacy problem we face tomorrow.

I would like to thank my technical advisor, David Evans, for providing sound mentorship and allowing me to attempt this thesis. Both my TCC advisors, Betsy Mendelsohn and Rosanne Simeone, deserve great thanks as well. They provided unsurpassed writing guidance; without them, my thesis would be incomprehensible. Finally, a thank you to Mark Boyns and Kirill Kouzoubov, their open source software contributed greatly to this thesis.

Table of Contents

Table of Figures

List of Tables and Examples

Abstract

Web services today, such as real time instant messaging and file sharing are insecure. This thesis implements an encryption program that has the capability of giving confidentiality to these services. The encryption program acts as a proxy server using modern encryption to secure all data transmissions of any Internet communication service. Thus, any application or service with proxy server support has the capability of securing its transmissions.

With an increasing reliance on digital data, the security of the digital communications medium has become more important. The sensitive information these digital lines carry can be privy to anyone without proper protection. This thesis fulfills a gap in the protection of newer, web communications using an encryption proxy. We describe the design using UML. Its implementation makes use of open source software and the Java Cryptographic Extensions to allow a robust, portable, and efficient encryption proxy.

1 Introduction

This thesis describes the securing of common web services such as instant messaging and file sharing through the creation of “Black Box”, an encryption proxy. These new modes of communication serve as primary conduits of contact for many people. However, these web services are extremely insecure. Protection of privacy is an important issue in society today. Consequently, the possibility of an intrusion of our privacy rises as well. Specifically, any type of digital communication makes an ideal target for information theft. Everyday Internet services, such as instant messaging and file sharing, are excellent examples of targets for information piracy. Attackers can tap into these Internet services easily to steal the information and secrets they carry. Securing the channels over which these services communicate provides the means to safeguarding them.

A proxy program acts as a go-between for applications accessing the Internet [1]. Black Box, being an encryption proxy, encrypts and decrypts all outbound and inbound Internet communication data. Thus, it secures the information communicated by any application or web service. Figure 1 further illustrates the proxy nature of the Black Box. In the figure, Black Box runs atop a computer operating system acting as a proxy between applications and the Internet. It encrypts all outbound, application data and decrypts all incoming, Internet data. Thus, following the figure from left to right, an application sends its data to Black Box. Next, Black Box encrypts the application data and sends it through the Internet to another computer system also running Black Box. Finally, the receiving Black Box proxy decrypts and hands the data to the appropriate application.

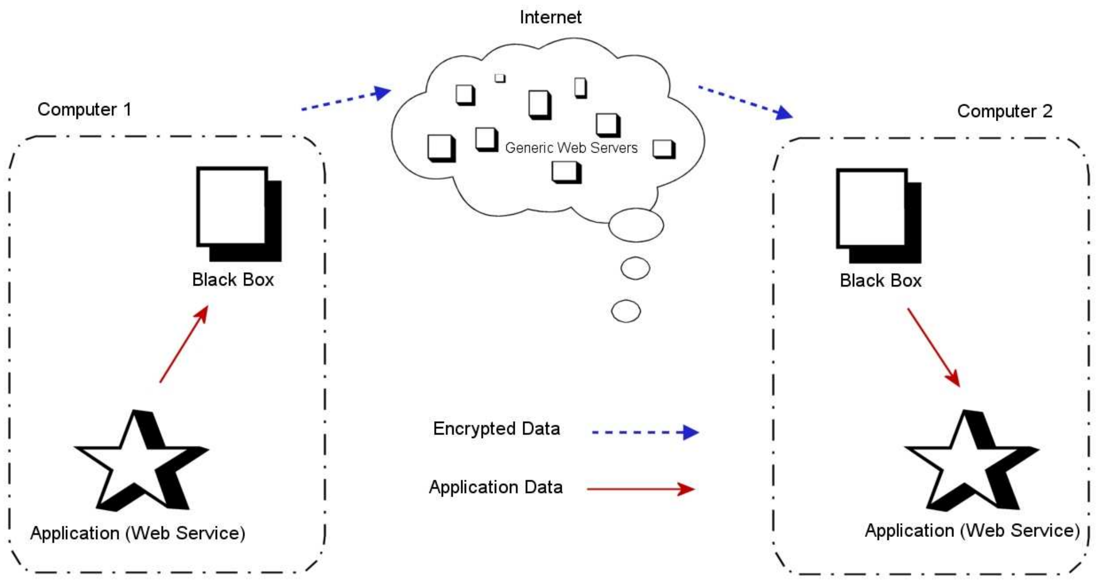

Figure 1. Program interaction

Black Box uses a symmetric block cipher known as TripleDES to encrypt. A cipher is an algorithm that changes plaintext (original text) into ciphertext (encrypted text) and vice versa. Today, ciphers protect digital communications with unrivaled security. Modern ciphers fall into two main categories; symmetric and asymmetric ciphers. Symmetric ciphers use one key to encrypt and decrypt messages. Conversely, an asymmetric cipher uses two keys; one for encryption and another for decryption.

The main problem with many ciphers is that a key exchange must take place before a secure channel is established. Key distribution is a problem that has plagued cryptographers for thousands of years. The problem involves distributing the keys safely to each user [2]. Oftentimes, great measures, such as personally delivering a key, must take place to ensure that an attacker does not compromise encrypted communication. For example, when using an asymmetric cipher, if an attacker poses as Alice and offers a fake public key to Bob, the attacker can continue to pose as Alice and intercept all information between the two. Security experts know this particular assault as the man-in-the-middle-attack [3].

Black Box makes use of a method known as the Diffe-Hellman key exchange for key distribution. In 1976, Whitfield Diffie and Martin Hellman presented a revolutionary solution to key distribution and introduced asymmetric ciphers in their paper, New Directions in Cryptography [10]. Diffie-Hellman allows two users to exchange a secret key without any prior agreement or knowledge over any medium. The paper propelled cryptography into the modern digital world allowing its use in new, exciting areas [4].

With the development of modern ciphers, programs began developing that provided encryption to the masses. Pretty Good Privacy (PGP) is such a program developed by Phil Zimmermann in 1991. PGP first uses RSA, an asymmetric cipher, to exchange a key and then turns to IDEA, a symmetric cipher, for encryption of any message given it. PGP does not make use of RSA entirely because RSA is computationally intensive and time consuming. Thus, PGP uses RSA only for its asymmetric nature and turns instead turns to a less intensive symmetric cipher for encryption, also known as session-key encryption [5]. People use PGP worldwide and its popularity continues to grow.

Zimmerman developed PGP to protect a new, rapidly expanding digital communication, email. He saw the growing need for protection of information privacy and the freedoms it entails in a digitally connected world. For without these protections, threatening organizations, governments, and entities could violate an individual's basic human right to privacy and secure communication. After PGP's development, Zimmerman became the centerpiece of a worldwide debate on the availability of strong encryption to the public. On one side of the debate, government and law enforcement argue that strong encryption allows criminals and terrorists to thwart legal wiretaps and communicate in secret. Directly opposing them, civil libertarians advocate that privacy is a fundamental human right as stated by article 12 of the Universal Declaration of Human Rights, "No one shall by subjected to arbitrary interference with his privacy … Everyone has the right to protection of the law against such interference or attacks." Civil libertarians argue that with the advent of digital communications, monitoring and wiretapping becomes easier. Consequently, this eavesdropping ability, unhindered by encryption, leads to threatening entities, such as the government or criminals, abusing it. Major corporations also ally with civil libertarians in this debate. Every major corporation employs encryption to protect their online transactions and private databases of corporate and client information. Without encryption, attackers can easily steal corporate secrets and cause major problems for a company. The debate rages on today, affecting the strength of encryption software such as PGP or this thesis's encryption proxy, Black Box [4].

With Black Box, users have the capability of securing their data transmissions saving them from potential information theft. These users include most of the average home users on the Internet today. Users of chat clients or file sharing clients can take advantage of the Black Box's capabilities. Thus, an encryption proxy adds protection to previously insecure digital communications easily and quickly.

Even with the availability of PGP, specific digital communications are still insecure. People use PGP primarily only with email. Zimmermann created it before the instant messaging and file-sharing phenomenon of today. Hence, society still needs a reliable, fast, and user-friendly program to secure these new channels of communication and ones to come. Some programs available today attempt to fulfill this need. However, they usually lack complete solutions or only provide encryption for their proprietary services. This thesis fills the gap in digital communications today with Black Box, an encryption proxy that works with all instant messaging and file-sharing software.

The rest of this thesis outlines modern cryptographic techniques and their history, explains the rationale behind the design decisions of Black Box, and reveals the implementation details of the proxy.

2 Background

For thousands of years, people have created ciphers to protect secrets. The ongoing war between cryptographers, those who develop ciphers, and cryptanalysis, those who break ciphers, drove this innovation. From simple courier mail to today's data transmissions, the success of securing messages with encryption has relied on the latest cryptographic research [4]. Thus, the designed encryption proxy takes advantage of all the major breakthroughs achieved by modern cryptography to provide a complete solution to securing web communications.

2.1 Encryption in Computer Networks

Computing experts have not always viewed encryption as the answer to computer network security. In 1968 Jack B. Dennis, a visionary professor from the Massachusetts Institute Technology, wrote a positional paper describing the basic requirements for successful and secure information networks. One of the three major requirements he listed was "the development of public, message-switched communications services so that adequate provisions are made for information security." He goes on to say that, "the security of messages sent between information systems via the internal public network is an important and serious problem …. Although message encryption is a useful technique where the communicating parties are able to make prior arrangements, the author is not convinced that this is workable in the context of public information services [6]." Thus, Dennis regarded message security as an important issue but did not identify encryption as a solution. Dennis further states that the problem with encryption stems not from the actual ciphers used but instead from key distribution. While his conclusions were valid for his time, they were made before modern breakthroughs in encryption made it a viable solution for overall network security.

2.1 Modern Cryptography

The Diffie-Hellman key exchange and RSA were two major breakthroughs that enabled modern encryption on today’s computer networks [4]. As mentioned in the introduction, Whitfield Diffie and Martin Hellman developed Diffie-Hellman in 1976. Published and described in their paper, New Directions in Cryptography, the protocol describes mechanisms that allow two users to agree upon a secret key without divulging any prior secrets over any insecure medium [3]. Diffie-Hellman relies on the erratic nature of one-way modular functions to negotiate a key exchange. One-way functions differ from two-way functions because one cannot easily reverse them knowing only the solution.

Let us further investigate Diffie-Hellman by examining an example key exchange between Alice and Bob. To start a Diffie-Hellman key exchange, both Alice and Bob use the function “Y^x (mod P)” and agree on a random value “Y” and prime “P.” Below, Table 2 lists the steps taken to arrive at an identical secret key.

Diffie-Hellman Key Exchange

1. Alice chooses a random value **“y”**
2. Alice sends “**Y^y (mod P)**” to Bob
3. Bob chooses a random value **“z”**
4. Bob sends **“Y^z (mod p)”** to Bob
5. Alice computes **“Y^{yz} (mod p)”** as **“(Y^y (mod p))^z mod p”**
6. Bob computes **“Y^{yz} (mod p)”** as **“(Y^z (mod p))^y mod p”**
7. They both arrive at the same final answer that they use as the secret key for symmetric encryption

Table 1. The steps in a Diffie-Hellman key exchange. Steps derived from [7].

In New Directions of Cryptography, Diffie and Hellman also discuss public-key cryptography. "*Public key distribution systems* offer a different approach to eliminating the need for a secure key distribution channel. In such a system, two users who wish to exchange a key communicate back and forth until they arrive at a key in common. A third party eavesdropping on this exchange must find it computationally infeasible to compute the key from the information overheard [8]." Diffie-Hellman was revolutionary in that it enabled a new class of encryption systems not requiring prearranged keys. RSA was ground-breaking for its use of public-key cryptography as well. However, unlike Diffie-Hellman, it encrypts messages instead of simply negotiating a key [4].

In 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman developed RSA, a cipher that would become a worldwide encryption standard [9]. Unlike many ciphers, RSA is asymmetric. RSA produces a pair of keys, public and private, for each user. Each key is unique and provides the means of either decrypting or encrypting messages. Let us examine an RSA communication to illustrate further. Alice and Bob wish to communicate securely using RSA. For Bob to send protected messages to Alice, Bob must first locate Alice's public key that she makes available to everyone. He then encrypts his message with Alice's public key. After delivery, Alice decrypts the message with the key that only she has privy to, her private key. For Alice's messages to Bob, the same method applies but backwards [4]. Figure 2 illustrates RSA pictorially. One might view RSA and Diffie-Hellman as a complete solution after this discussion. However, a problem remains that plagues most modern encryption schemes: the man-in-the-middle attack.

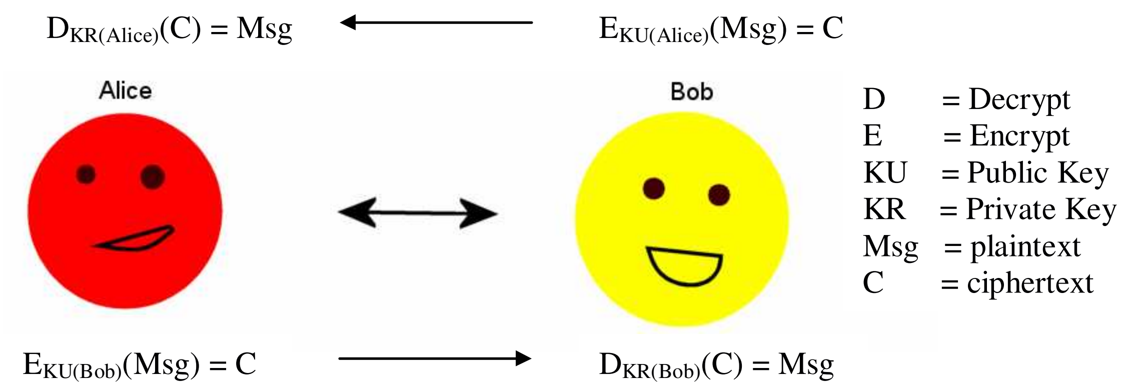

Figure 2. Alice and Bob use RSA

As mentioned in the introduction, the man-in-the-middle attack involves an attacker intercepting the exchange of public keys between two individuals, introducing his or her own public key to each, and then intercepting all encrypted messages between the two. Thus, the two communicating users encrypt their messages with the attacker's public key allowing the attacker to read their messages after interception. After the attacker finishes reading the intercepted message, he or she encrypts the message with the public key of the intended recipient. The corresponding individuals can then never detect the attack if the attacker executes it flawlessly [3].

Today, encryption software uses partial solutions that minimize the chances of a person-in-the-middle attack. The most widely used solution is a digital certificate. Digital certificates contain an individual's public key encrypted with the private key of a trusted source. Thus, a trusted source, usually a government agency or large company, vouches for an individual. After receiving a digital certificate, one retrieves the embedded key by decrypting the certificate using the trusted source's public key. The trusted source's public key is widely available from multiple sources, and thus trustworthy. Thus, the only risk present in this security model is the trusted source's private key. If an attacker compromises its private key, all its issued certificates become useless [5].

The design of a complete, secure encryption proxy required understanding and use of all the concepts presented above. For example, Black Box performs session-key encryption by utilizing Diffie-Hellman for initial key agreement, and then TripleDES for symmetric encryption. However, the novelty of Black Box lies not in the technology it employs but the work it performs. Thus, these technical details provide the means to an end. This thesis attempts innovation not through new discoveries in cryptography but rather by finding a new, novel application of encryption. Like PGP, Black Box secures widely used digital communications that currently lie susceptible to attack.

3 Design

We utilized the Universal Modeling Language (UML) to design Black Box. UML's benefits include its graphical notation, heavy use of object-oriented design, program flow modeling, intuitive feel, and observations into the usage of objects by users and systems. Software designers use UML to model software systems during the design process to better understand and implement their programs. UML conveys the programmer's design in terms of use, structure, and flow [10]. Black Box also incorporated open source software in its implementation. Accordingly, the structure of the open source software greatly affected the overall design.

3.1 Including Open Source Code

Black Box utilizes open source software to provide the functionalities of a Hypertext transfer protocol (HTTP) and SOCKS proxy server. Both the HTTP and SOCKS protocols outline the interactions between a proxy server and the clients it serves. Without these protocols, a standard would not exist for implementing universal proxy support within software. Both protocols are non-proprietary, free, and available on the web.

Likewise, open source software shares these same aspects. Open source software includes any program that a developer has specifically licensed to be freely available. Consequently, programmers have the option of making their source code obtainable to others under specific software licenses. These licenses vary greatly in terms of reusing protected source code in other programs. Restrictions on reuse usually depend on whether the borrowed code appears in open source or proprietary software. Independent developers often lean towards allowing reuse only in other open source software. In contrast, software companies favor letting open sourced code appear in proprietary programs.

The open source software Black Box uses allows for modification and use in resulting open source software only. Thus, Black Box itself falls under the GNU General Public License (GNU GPL) that allows for modification of GPL code in freely distributed software only [11]. GNU itself is a project launched in 1987 that works towards a complete UNIX-like environment in the form of free software.

Overall, the use of open source HTTP and SOCKS software greatly reduced the time and effort needed to create an overall design strategy. The functionality of the open source software allowed the design of Black Box to focus on its main task, encryption. Accordingly, the resulting UML design reflects the usage of open source HTTP and SOCKS proxies.

3.2 UML Class Diagram

The UML class diagram communicates the structure of the software system using object-oriented (OO) design. Thus, the UML class diagram uses classes, the primary building block used in OO design to group ideas, as its basis. OO design views classes as the description of the variables and methods inherent within similar types of objects. A class can model real-life objects according to the natural data and operations present in all items. Consequently, OO design designates a particular class instantiation as a specific object of that class [10]. To illustrate these concepts take the following example. A programmer decides to make an analog clock program. The programmer creates a main clock class containing the necessary data needed, mainly time and its subdivisions of hours, minutes, and seconds, and operations on time, such as start time or reset time. The programmer can further divide the physical clock into its components, the hour, minute, and second hand. This division of the clock into its physical components creates an aggregation in OO terms. Finally, once the programmer starts the program, it instantiates the clock class into a clock object. Figure 3 contains the class diagram for the example analog clock. The dotted arrows represent dependencies between the classes.

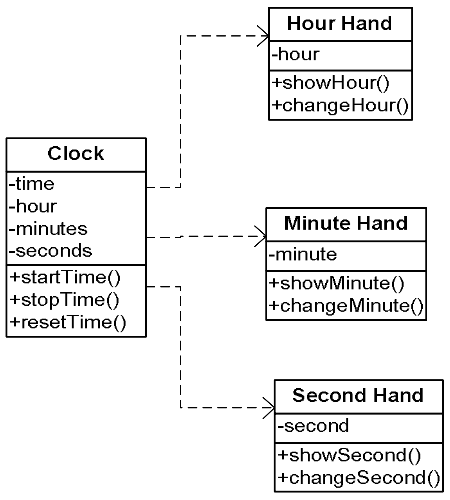

Figure 3. UML Class Diagram of example analog clock

Dependency arises when one class subclasses another. A solid arrow depicts inheritance, another OO feature that represents the derivation of a class from a similar one.

Figure 4 illustrates the class diagram containing the nine classes used within Black Box. The proxy class diagram contains three main classes, the *HTTPProxyServer, SOCKSProxyServer*, and *BlackBoxGUI* classes. Both the *HTTPProxyServer* and *SOCKSProxyServer* classes represent open source software used to perform the functions of a HTTP and SOCKS proxy. The two classes in the diagram abstract the open source software into objects to incorporate them into the Black Box design. The *HTTPProxyServer* class handles incoming HTTP proxy connections, encryption configuration, and starting the server. Consequently, the *HTTPProxyServer* class starts the entire server running and creates *SOCKSProxyServer* and *BlackBoxGui* objects. The *SOCKSProxyServer* class represents a thread that handles all SOCKS proxy connections. A thread executes a differing program flow concurrently with the main program. Thus, when a program needs simultaneous execution of differing code, it uses threads. Within Black Box, two threads concurrently handle possible HTTP and SOCKS connections. The *BlackBoxGui* class executes as a thread that handles all the encryption functionalities of Black Box. These functionalities include configuring encryption, connecting to perform key agreement, and actual encryption and decryption of data.

Six classes help the three main classes perform their tasks. Both the *HTTPConnection* and *SOCKSConnection* classes represent the connections either a HTTP or SOCKS proxy handles. They inherit from a general *Connection* class that abstracts a connection into its describing data and operations. Connection data includes socket as well as input and output stream information. A socket represents a computer connection point, analogous to a telephone or antenna socket. Input and output streams characterize the flow of data from a socket or, more generally, any object. Connection operations consist of reading, writing, and closing a socket. Thus, by inheriting from the *Connection* class the *HTTPConnection* and *SOCKSConnection* classes receive all the data and operations of a general connection. Further, they each contain specific data and operations respective to their specialized type of connection. For instance, the

HTTPConnection class creates data objects that stand for the HTTP request and replies received as stated by the HTTP protocol. Likewise, the *SOCKSConnection* class has data representing a SOCKS request and reply.

The *EncryptListner* class represents a thread listening for connections made to establish key agreement by other remote encryption proxies. *BlackBoxGui* instantiates an *EncryptListner* thread after the user starts encryption. In turn, the *EncryptListner* thread creates an *Encrypt* object. The *Encrypt* class encapsulates the cryptographic functions of encryption and decryption. Thus, when a connection wants to decrypt or encrypt its data, it uses an *Encrypt* object. The *Encrypt* class inherits from the *EncryptionConnection* class. The *EncryptionConnection* class represents all the cipher algorithms, keys, and data flows associated with encryption. Thus, through inheritance, the *Encrypt* class manages all the logistics of encryption.

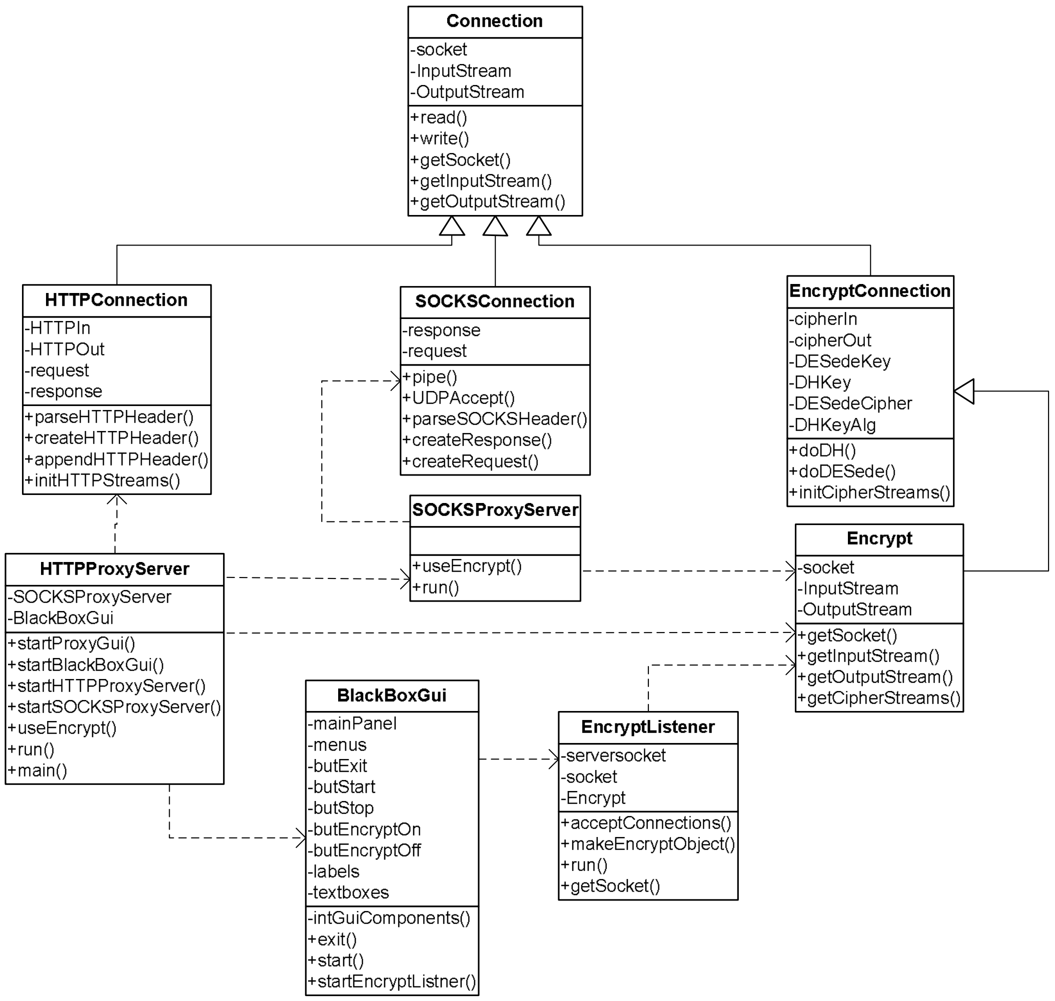

Figure 4. UML Class Diagram for Black Box

4 Data Transfer and Encryption

To better understand the operation of Black Box, this section examines how two Black Box programs communicate, negotiate a secret key, and transfer secure data. Applications on a local computer access Black Box through local HTTP and SOCK connections. HTTP and SOCK headers state what server an application requests. Consequently, Black Box parses these headers to find the correct server to establish a connection with on behalf of the application. Figure 5 depicts SOCKS or HTTP encoded application packet. Black Box then establishes a connection and handles all the information transferred between the application and its requested servers. This describes the normal proxy behavior of Black Box.

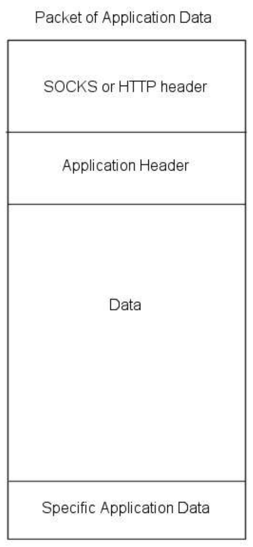

Figure 5. Application data packet with SOCKS or HTTP header

For encryption, a remote computer running Black Box negotiates a secret key with the local computer through a direct connection. Afterwards, the local computer uses Black Box to encrypt all received application data before transferring the data out to requested servers. Eventually, the requested servers relay this data to the remote computer that shares the negotiated secret key, thus allowing it to decrypt the application data. Figure 6 outlines the interaction between a local and remote computer running Black Box.

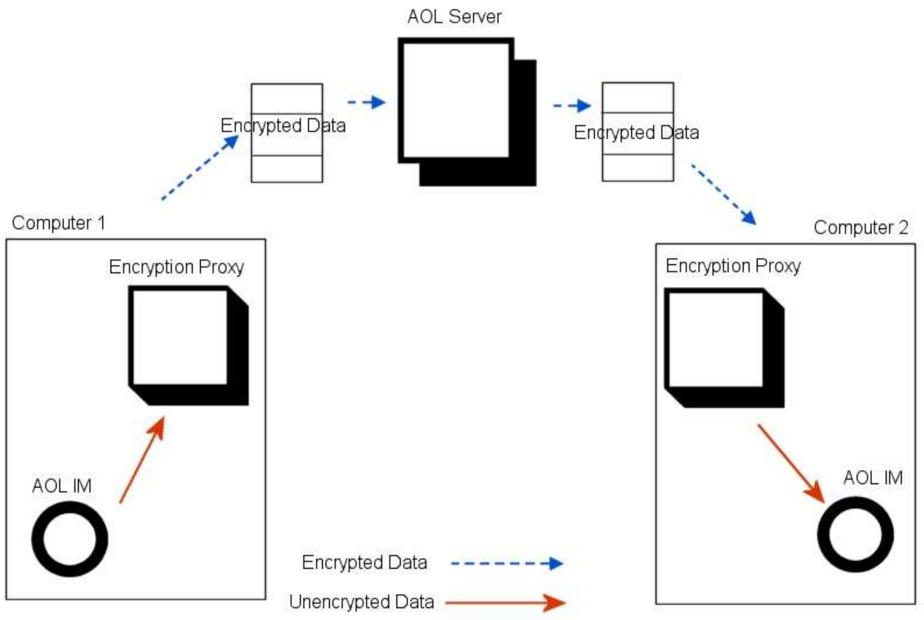

Figure 6. Black Box encrypting data between two computers

Black Box encryption relies on the fact that users use the same application on both sides of the encrypted connection. It also assumes that the applications use a protocol that allows for encryption. For example, a user uses an instant messaging application that talks with its server and correspondents using a specialized protocol. If Black Box encrypts the entire application data packet before sending it to the server, the server will not understand the encrypted message and will not correctly forward it to the intended recipient. Thus, the Black Box must make note of the applications and their protocols to encrypt data properly.

Black Box can gain this knowledge of an application's protocol either by dynamically loading it as a configuration file or hard coding the protocol in the source code. These configuration files would list the application's protocol and how and where to encrypt the application's data. Hard coding the application would achieve the same goal. This version of Black Box utilizes hard coding instead of configuration files to gain knowledge of a particular application's protocol.

Black Box's ability to encrypt relies on the assumption that an application's developer releases their proprietary software protocol. Recent trends show that software companies generally disclose their application's protocols [17]. However, one exception is America Online (AOL). The company does not disclose the Open System for Communication in Real Time (OSCAR) protocol for its popular AOL instant messaging client (AIM) to the public. However, some people reversed engineered AOL's AIM protocol and currently publish it as the Fake AIM (FAIM) protocol on the Internet [18].

Black Box manages to encrypt AIM messages by using FAIM to partially parse AIM application packets and encrypt the text string within. However, FAIM does fail in some areas. FAIM contains inaccuracies regarding the structure of AIM message packets as well as a few other AIM commands. Consequently, this implementation of Black Box required reverse engineering parts of AIM's protocol, and thus, this process increased overall development time.

5 Implementation

Java, the chosen language, made building Black Box an easier and more efficient task overall. Likewise, the use of open source software cut overall development time by eliminating the need to implement HTTP and SOCKS proxies. However, one must take great care whenever reusing code. Detailed knowledge of the reused code proves necessary because incorrect application usually leads to software bugs and unpredictable program behavior. Lastly, using the Java Cryptographic Extensions (JCE) helped simplify the use of encryption by abstracting and encapsulating complex cipher algorithms into classes.

5.1 Usage of Java

Several requirements influenced the choice of Java as the main language for this project. Black Box needs a small overall size for easier distribution across limited bandwidth. The project required a quick, efficient design and implementation of the Black Box. Lastly, Black Box needs to work across a variety of platforms including Windows, Linux, and UNIX.

Using Java created a smaller, more efficient encryption proxy. Small applications shorten download times across connections with limited bandwidth including modems. By making Black Box smaller, its availability increases to include more of the public. A small size also makes Black Box more transportable since it fits on floppy or any other removable disks. Furthermore, smaller programs usually execute more quickly and efficiently.

Java’s design as a programming language allows for quick design and implementation of software projects. Through its object-oriented approach, strong type checking, and error catching capabilities, Java lets programmers translate their software designs into reliable code. If a programmer designs a project in the object-oriented fashion of the Universal Modeling Language (UML), Java automatically lends itself to rendering the code easily. Thus, this project took advantage of Java’s features for a quick design and implementation of Black Box.

Lastly, Java’s design as a write once, run anywhere language finalized its choice as this project’s main programming language. The cross-platform capabilities derived from using Java allow a program to gain acceptance with a wider base of users. Thus, the users of Windows, Linux, and UNIX can use Black Box interactively with each other [12].

Overall, Java allowed this project to meet its encryption proxy requirements in less time and more efficiently. Java’s excellent characteristics as a portable, easy to use, and efficient language proved valuable to this thesis.

5.2 Usage of Java Cryptographic Extensions

The JCE aided development by simplifying the usage of complex encryption algorithms within Black Box. The JCE eased implementation by providing an easy to use interface, thorough documentation, and an abstraction of complex cryptographic details. It includes Java implementations of both encryption algorithms used in Black Box, the Diffie-Hellman key exchange and Triple Data Encryption Standard

(TripleDES). Also, the JCE's inclusion into the Java 2 SDK v 1.4, allows any machine with an up-to-date installation of Java to run the encryption routines in Black Box.

Using JCE greatly simplifies using complex encryption algorithms because of the automatic key generation facilities and intuitive cipher interfaces JCE provides. To use any cipher to decrypt and encrypt data using JCE, one first creates a cipher object, naming what encryption algorithm to use, and then initializes the parameters for the algorithm. The example below demonstrates the ease of development that JCE provides through abstraction when using TripleDES.

```
// Create a TripleDES cipher object
Cipher 3DEScipher = Cipher.getInstance("TripleDES");

// Initialize the cipher object to encrypt and the user generated key
3DESCipher.init(Cipher.ENCRYPT_MODE, 3DESkey);

// Create clear text to encrypt
byte[] cleartext = "Test with some text".getBytes();

// Encrypt the clear text into cipher text
byte[] ciphertext = AESCipher.doFinal(cleartext);

// Initialize the cipher object to decrypt now
3DESCipher.init(Cipher.DECRYPT_MODE, 3DESkey);

// Decrypt the cipher text into clear text
byte[] dcleartext = 3DESCipher.doFinal(ciphertext);
```

Example 1. JCE Encryption Abstraction

Without this abstraction of the encryption algorithm, the development time for Black Box increases.

The key generation facilities of the JCE remove the complex details of creating a random key from a perfectly random seed. Oftentimes, in cryptography, an attacker sees the secret key as a weak point rather than the actual encryption algorithm itself. This fact places great importance on generating a random key. To create a key, one simply creates a key generator object, initializes the generator object, and generates the key. Example 2 below demonstrates abstraction of key generation.

```
// Create an key generator object
KeyGenerator 3DESkeyGenerator = KeyGenerator.getInstance("TripleDES");

// Initialize the key object with a source of randomness and keysize
3DESkeyGenerator.init(RANDOM_SEED, 512);

// Generate the key and place it into a secret key object
SecretKey 3DESkey = 3DESkeyGenerator.generateKey();
```

Example 2. JCE Key Abstraction

JCE provides facilities to use encryption effectively within Java applications. It allows developers to concentrate on the use and incorporation of encryption rather than its implementation. JCE provided this project with valuable tools to decrease total development time of Black Box [13].

5.3 Implementing Encryption

The encryption algorithms chosen for Black Box as mentioned above were Diffie-Hellman and TripleDES. Usage of both ciphers resulted in session-key encryption. Thus, Black Box makes a direct connection with a remote proxy to perform Diffie-Hellman and generate a secret key. Black Box employs a 1024-bit Diffie-Hellman key size using SKIP (Simple Key Management for Internet Protocols) as its standard modulus, "Y", and base, "P", in the standard equation Y^x (mod P). Standard Diffie-Hellman interactions must occur between a local and remote Black Box program to establish an identical secret key. After Diffie-Hellman, Black Box initializes a 168-bit secret key for TripleDES. A key size of 168-bits provides moderate security on modern computer networks [14]. The network nature of Black Box necessitates the initialization of TripleDES, orginially a block cipher, as a stream cipher. Thus, instead of encrypting 64-bit blocks of data, the proxy instead encrypts 8-bit blocks. This initialization proves useful in streaming network applications, such as Black Box, because applications usually do not read or write 64-bit blocks on a network.

Black Box uses the TripleDES block cipher instead of the originally proposed AES cipher because of the TripleDES support in JCE. Since JCE's inclusion into Java v1.4, this support essentially ensures Black Box runs properly on machines that support Java. The JCE specification does list AES as a supported cipher; however, its implementation in the JCE is incomplete [13]. Overall, deploying encryption proved not difficult because of the classes provided by JCE.

5.4 Usage of Open Source Software

Black Box utilizes two open source software packages, Muffin, a HTTP proxy server, and Java SOCKS Server, a SOCKS proxy server. Both packages are freely available with source code via the Internet. Black Box uses their code primarily because it is robust, efficient, and concise. Mark Boyns, a student at San Diego State University, developed Muffin as an HTTP proxy with the capabilities of filtering and caching web pages [15]. Kirill Kouzoubov, a student at the University of South Australia, developed both the Java SOCKS Server and SOCKS library for future use by other developers [16].

Black Box combines both open source proxies. This allows it to accept both HTTP and SOCKS proxy connections. It then adds encryption capabilities to both types of connections by implementing the classes described in section 3.0. For its main GUI, Black Box uses Muffin because of its superior HTTP connection monitoring. Users can also configure the HTTP, SOCKS, and encryption connections from the main GUI. Both open source programs provided little documentation about their design and implementation. Thus, acquiring a deep understanding of both took a great deal of development time and experimentation. However, in general the usage of open source code cut proxy development time and allotted a greater amount of time towards development of encryption.

6 Conclusion

This chapter splits the conclusion into three sub-sections, the summary, interpretation, and recommendation section, so that this thesis presents a more comprehensive conclusion. First, the summary section provides an objective and definite review of Black Box and its development. Next, the interpretation section draws conclusions about Black Box and knowledge gained for its development. Finally, the recommendations section makes suggestions of how to further enhance Black Box and investigate new encryption problems.

6.1 Summary

The final version of Black Box for this thesis includes:

- proxy support for multiple HTTP, HTTPS, SOCKSv4, and SOCKSv5 connections
- session key encryption using Diffie-Hellman and TripleDES
- support for encryption of AIM messaging between two parties
- support for multiple platforms including UNIX, Windows, OSX, and Linux

This thesis utilized Java to create an encryption proxy named Black Box. Thankfully, the addition of JCE into Java 2 SDK v1.4 specification ensures the support of strong encryption routines on any system with an updated Java Runtime Environment (JRE). Thus, Black Box runs with encryption on any machine with an up-to-date JRE. However, Black Box's encryption capabilities require specific knowledge of the encrypted data's application protocol. The final implementation of Black Box only includes the AIM protocol, and thus, the proxy only secures this service.

Utilizing open source, Black Box is itself open sourced and freely available under the GNU GPL. The Black Box's design reflects the use of open source through an accurate UML Class description that relay the proxy's underlying structure and flow.

6.2 Interpretation

The design of Black Box includes a comprehensive UML Class diagram. This diagram allows future developers to understand and modify the existing source code with ease. Thus, future versions of this Black Box or programs based of its source code have a greater chance of existence.

The Black Box's inclusion of open source software permitted a fast, robust, and efficient implementation cycle. The use of open source allowed the focus of development to shift towards important issues including the implementation of working encryption and inclusion of the AIM/OSCAR protocol. In particular, development focused on including the OSCAR protocol because of AIM's massive popularity and OSCAR's undisclosed nature. The use of open source software also directly affected the availability of Black Box under the GNU GPL, and thus, made it free software.

Written in Java, Black Box runs on several platforms. Its use of JCE also allows it encryption capabilities to be portable as well. Thus, with this type of support, Black Box ensures that not only will Windows users deploy it, but also UNIX, Linux, and OSX users as well.

The strong proxy connection support of Black Box ensures its compatibility with a wide range of proxy capable applications. Thus, any application that supports either SOCKS or HTTP proxy connections has the capability of encrypting all its network bound data using Black Box. This added encryption capability presents strong implications for securing a wide range of previously insecure data transmissions and communications. Thus, since Black Box can secure web applications, such as AIM, it secures a channel of digital communication millions use everyday.

Overall, Black Box strives to meet its original intent of securing all instant messaging and file sharing services. However, the final version of Black Box secures only the AOL instant messaging service. This one service presented a challenge because its application protocol is proprietary and undisclosed. Hence, the thesis attempts to secure the most difficult service available. If one views the final version of Black Box in this light, one clearly sees that it is a success. Black Box proves the possibility of securing all web services by securing the most challenging service first. Thus, application of Black Box to other web services is a strong possibility.

6.3 Recommendations

The final implementation of the Black Box leaves many avenues open for development. Most importantly, the addition of new application protocols to Black Box is a necessity. This addition ensures the thesis's original intention of securing all instant messaging and file-sharing services. Since most of the services have open, disclosed protocols, one can easily add support for new web services. Coinciding with the addition of application protocols is the development of an application detection engine. Thus, when a particular application connects, Black Box chooses the appropriate application protocol.

Another important feature enhancement is the customization and improvement of the encryption capabilities of the program. To improve the encryption capabilities of the proxy, one could research new cryptographic findings and explore how to incorporate them into Black Box. For instance, the usage of public key rings, password authentication, or digital certificates could provide a more secure implementation of Black Box. Customization of encryption within Black Box could involve letting users choose which encryption algorithm they wish to use to secure a service.

The future enhancements that one can add to Black Box are limitless. However, all added features need careful consideration and study on how they affect the overall security. One can greatly affect a secure application by adding new features that present inherent weaknesses to the software's security model. Such feature might include saving keys unencrypted, badly implemented authentication schemes, or poor use of passwords.

7 Works Cited

[1] Angel, Jonathon. Proxy Servers. Network Magazine, 1 April 1999. 7 September 2001 <http://www.networkmagazine.com/article/NMG20000724S0061>.

[2] Knudsen, Jonathon. Java Cryptography. Cambridge: O'Reilly & Associates, Inc., 1998.

[3] Pirooz, Hamid. Diffie-Hellman Public Key Distribution Scheme: A Complete Overview. Sans Institute, 4 December 2001. 8 September 2001 <http://www.sans.org/infosecFAQ/encryption/diffie.htm>.

[4] Singh, Simon. The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography. New York: Anchor Books, 1999.

[5] "The International PGP Home Page." 15 September 2001 <http://www.pgpi.org>.

[6] Dennis, Jack B. "A Position Paper on Computing and Communications." Communications of the ACM. 2 (1968): 370-377.

[7] Garrett, Paul. Making, Breaking Codes: An Introduction to Cryptology. Upper Saddle River: Prentice Hall, 2001.

[8] Diffie, Whitfield, and Martin B. Hellman. New Directions in Cryptography. IEEE Transactions on Information Theory: 1976.

[9] RSA Security Inc. "RSA Laboratories Frequently Asked Questions about Today's Cryptography 4.1." 8 September 2001 <http://www.rsa.com/rsalabs/faq/index.html>.

[10] Schmuller, Joseph. Sams Teach Yourself UML in 24 Hours. Indianapolis: SAMS, 1999.

[11] "GNU's Not Unix! – the GNU Project and the Free Software Foundation (FSF)." 15 March 2002 <http://www.gnu.org>

[12] "java.sun.com." 8 January 2002 <http://sun.java.com>

[13] "Java™ Cryptographic Extensions (JCE)." 10 January 2002 <http://java.sun.com/products/jce>

[14] Garms, Jess and Daniel Somerfield. Professional Java Security. Birmingham: Wrox Press, 2001.

[15] "MUFFIN.DOIT.ORG." 156 January 2001. <http://muffin.doit.org>

[16] "Java SOCKS Proxy Server." 16 January 2002. <http://jsocks.sourceforge.net>

[17] "IETF Internet Engineering Task Force." 17 February 2002 <http://www.ietf.org>

[18] "FAIM/AIM/OSCAR protocol." 17 February 2002 <http://www.oilcan.org/oscar>

8 Glossary of Terms

Advanced Encryption Standard (**AES**) – An encryption algorithm designated for use in encryption by the United States. The National Institute of Standards and Technology choose Rijndael as this cipher and supplant DES.

AOL Instant Messenger (AIM) – A popular instant messaging program.

America Online (AOL) – A large software company and Internet Service Provider (ISP).

asymmetric cipher – A cipher that uses two keys, one to encrypt and the other to decrypt.

block cipher – A cipher that encrypts blocks of plaintext at a time.

cipher – A general name given to any algorithm or method that changes plaintext into ciphertext through encryption. A cipher involves a key to perform the encryption and decryption.

ciphertext – A message after being obscured by encryption.

concurrent program – A program that has concurrent independent tasks or threads of control.

cryptography – The science of hiding or encrypting a message.

cryptology – The science of secrets including cryptography and cryptanalysis.

cryptanalysis – Unscrambling ciphertext without a key.

decryption – The process of deciphering or decoding a ciphertext back into plaintext.

Data Encryption Standard (**DES**) – An encryption algorithm adopted by the United States in 1976 for encryption of all its data. Today, the United States uses AES instead.

Triple DES (3DES, DESede) – An encryption algorithm that uses three rounds of DES to increase the key bit size.

Diffie-Hellman key exchange (Diffie-Hellman) – A protocol by which two parties can establish a secret key without sending any sensitive data. The two parties can then use the secret key with a symmetric cipher for encryption. Whitfield Diffie and Martin Hellman created Diffie-Hellman in 1976.

Fake AIM (FAIM) – The unofficial AIM/OSCAR protocol.

GNU – A project launched in 1984 to provide a complete UNIX-like environment as free software. GNU is a recursive acronym for ``GNU's Not Unix''. It is pronounced "guh-NEW".

GNU General Public License (GPL) – A particular software license developed by GNU that promotes free software and source code modification.

key – Special knowledge or information that allows one to encrypt or decrypt ciphertext using an encryption algorithm or cipher. Security experts regard keys as the vulnerable point in encryption, not the encryption algorithm itself.

key distribution – The process of distributing a key for encryption and decryption by multiple parties of two or more.

encryption – The process of encrypting or hiding the meaning of a message.

man-in-the-middle attack – An attack that renders modern asymmetric encryption useless.

multi-threaded program – A program that has two or more threads of execution. Allows for the overlapping or concurrent processing of several tasks.

object-oriented design – A software design paradigm that promotes viewing computation as the interaction among semi-independent objects that each contain its own internal state and operations to mange that state.

Open System for Communication in Real Time (OSCAR) – The official undisclosed AIM protocol.

Pretty Good Privacy (PGP) – A computer encryption algorithm used to encrypt and decrypt digital data using session-key encryption. PGP uses a combination of RSA and any symmetric cipher and its primary use is with email.

plaintext – A message before encryption.

private key – The key used with asymmetric ciphers to decrypt ciphertext or digitally sign a message.

proxy server – A server or agent that allows two parties to communicate through it. A proxy can perform actions on the information flowing through it such as encryption, caching, or filtering.

public key – The key used with asymmetric cipher to encrypt plaintext or verify a digitally signed message.

public key cryptography – A system of cryptography that utilizes asymmetric ciphers to overcome key distribution problems.

Rijndael – A block cipher developed by Joan Daemen and Vincent Rijmen and designated as AES.

RSA – An asymmetric cipher invented by Ron Rivest, Adi Shamir, and Len Adleman at MIT in 1977.

session-key encryption – The process of using an asymmetric cipher to establish a symmetric cipher's key with multiple parties. One then uses the established key for any further encryption.

symmetric cipher – Any cipher that uses one key to encrypt and decrypt.

thread – An independent path of execution (task) within a program. Threads occur within concurrent or multi-threaded programming.

Universal Modeling Language (UML) – A descriptive software and system modeling language. It consists of subsets of graphical elements combined into diagrams to convey functional and structural data. The resulting diagrams represent multiple views of a single model. Grady Booch, James Rumbaugh, and Ivar Jacobson developed UML throughout the 1980's until they formalized it in 1997.