University of North Florida
UNF Digital Commons

UNF Graduate Theses and Dissertations

Student Scholarship

2007

Performance Analysis of Security Protocols

Praveen Kumar Donta
University of North Florida

Follow this and additional works at: https://digitalcommons.unf.edu/etd

Part of the Computer Sciences Commons

Suggested Citation

Donta, Praveen Kumar, "Performance Analysis of Security Protocols" (2007). *UNF Graduate Theses and Dissertations*. 172.
https://digitalcommons.unf.edu/etd/172

This Master's Project is brought to you for free and open access by the Student Scholarship at UNF Digital Commons. It has been accepted for inclusion in UNF Graduate Theses and Dissertations by an authorized administrator of UNF Digital Commons. For more information, please contact Digital Projects.
© 2007 All Rights Reserved

PERFORMANCE ANALYSIS OF SECURITY PROTOCOLS

by

Praveen Kumar Donta

A project submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

April, 2007

Copyright (©) 2007 by Praveen Kumar Donta

All rights reserved. Reproduction in whole or in part in any form requires the prior written permission of Praveen Kumar Donta or his designated representative.

The project "Performance Analysis of Security Protocols" submitted by Praveen Kumar Donta in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by:	Date
Signature deleted Dr. Sanjay Ahuja Project Director	4/9/07
Signature deleted Dr. Charles N. Winton Graduate Director	4/10/07
Signature deleted Dr. Judith L. Solano Director, School of Computing	4/10/07

ACKNOWLEDGEMENTS

I am thankful for Dr. Sanjay Ahuja's invaluable suggestions and assistance throughout this project. My wife, Sunitha, and my son, Nihal, gave their help and encouragement during long hours of project work. My gratitude to my brother, Ramesh, for his continuous support in every part of my life. I am thankful for my parents' care and love.

CONTENTS

List of Figures viii

List of Tables x

Abstract xi

Chapter 1: Introduction 1

Chapter 2: Security Algorithms, Message Digest and SSL 3

2.1 Symmetric Key Algorithms 3

2.1.1 DES 3

2.1.2 3DES 3

2.1.3 AES 4

2.1.4 RC4 4

2.2 Public-Private Key Algorithms 5

2.2.1 RSA 5

2.2.2 ElGamal 5

2.3 Message Digests 6

2.3.1 MD5 6

2.3.2 SHA1 7

2.4 SSL 7

Chapter 3: Project Description 9

3.1 Overview 9

3.2 Hardware 11

3.3 Software 11

Chapter 4: Testing Methodology 12

4.1 Security Algorithms Testing Method...12

4.2 SSL Testing Method 12

Chapter 5: Results 13

5.1 DES Versus RC4 Encryption and Decryption Times 13

5.2 3DES Versus RC4 Versus AES Encryption and Decryption Times
With 192 Bit Key 15

5.3 RC4 Versus AES With 5K File 17

5.4 RC4 Versus AES With 10K file 19

5.5 RSA Algorithm Encryption and Decryption...21

5.6 ElGamal Encryption and Decryption Times...24

5.7 Digital Certificate SHA1 Versus MD5 With RSA 27

5.8 SSL Communication Timings for Two Sets of Security Algorithms...........28

Chapter 6: Analysis and Conclusions 30

6.1 Analysis of Test Results..30

6.2 Security Algorithms Performance Comparison From Previous Studies.........32

6.2.1 Crypro++ 5.2.1 Benchmarks 32

6.2.2 Testing a Variety of Encryption Technologies 34

6.2.3 Security Performance 35

6.3 Conclusions...37

References.. 38

Appendix A: Security Protocols Code Listings 39

Appendix B: SSL Code Listings 111

Vita 120

LIST OF FIGURES

Figure 1: Example of a Digital Signature Scheme 6

Figure 2: Layers for a User Browsing With SSL 8

Figure 3: Simplified Version of the SSL Connection Establishment Protocol 8

Figure 4: DES Versus RC4 With 64 Bit Key Encryption and Decryption 14

Figure 5: 3DES Versus RC4 Versus AES Encryption With 192 Bit Key 15

Figure 6: 3DES Versus RC4 Versus AES Decryption With 192 Bit Key.......................16

Figure 7: RC4 Versus AES Encryption With 5K File 18

Figure 8: RC4 Versus AES Decryption With 5K File..18

Figure 9: RC4 Versus AES Encryption With 10K File 20

Figure 10: RC4 Versus AES Decryption With 10K File...20

Figure 11: RSA Encryption Times for 100 bytes File 22

Figure 12: RSA Encryption Times 22

Figure 13: RSA Decryption Times for 100 bytes File...23

Figure 14: RSA Decryption Times 23

Figure 15: ElGamal Encryption Times for 100 bytes File...25

Figure 16: ElGamal Encryption Times 25

Figure 17: ElGamal Decryption Times for 100 bytes File 26

Figure 18: ElGamal Decryption Times...26

Figure 19: Digital Certificate Times SHA1 Versus MD5 With RSA...............................27

Figure 20: Digital Certificate Maximum Throughput SHA1 and MD5 With RSA 28

Figure 21: SSL Communication 3DES-SHA1 Versus RC4-MD5 With RSA Maximum Throughput 29

Figure 22: Maximum Throughput of Secure Socket Layer 37

LIST OF TABLES

Table 1: Throughput For Security Algorithms Using Crypto++ 33

Table 2: Performance of RSA Public Key Algorithm Using Crypto++ 34

Table 3: Testing a Variety of Encryption Technologies... 35

Table 4: Digital Signature Verification Times and Maximum Throughput 36

ABSTRACT

Security is critical to a wide range of applications and services. Numerous security mechanisms and protocols have been developed and are widely used with today's Internet. These protocols, which provide secrecy, authentication, and integrity control, are essential to protecting electronic information.

There are many types of security protocols and mechanisms, such as symmetric key algorithms, asymmetric key algorithms, message digests, digital certificates, and secure socket layer (SSL) communication. Symmetric and asymmetric key algorithms provide secrecy. Message digests are used for authentication. SSL communication provides a secure connection between two sockets.

The purpose of this graduate project was to do performance analysis on various security protocols. These are performance comparisons of symmetric key algorithms DES (Data Encryption Standard), 3DES (Triple DES), AES (Advanced Encryption Standard), and RC4; of public-private key algorithms RSA and ElGamal; of digital certificates using message digests SHA1 (Secure Hash Algorithm) and MD5; and of SSL (Secure Sockets Layer) communication using security algorithms 3DES with SHA1 and RC4 with MD5.

Chapter 1

INTRODUCTION

There are many network security algorithms widely available today for use, including symmetric-key algorithms, public-private key algorithms, and secure hash functions. The purpose of this project was to determine which algorithm performs better for given input data and to conduct a performance analysis of various security algorithms.

Symmetric key algorithms use the same key for encryption and decryption. Some of the symmetric algorithms are DES (Data Encryption Standard), 3DES (Triple DES), RC4 and AES (Advanced Encryption Standard). In public key algorithms, the encryption and decryption keys are different. It is not feasible to derive the decryption key from the encryption key. Some of the public key algorithms are RSA and ElGamal. Secure hash functions or message digests work on an authentication scheme and do not require encrypting the entire message. A secure hash function scheme is based on the idea of a one-way hash function that computes from an arbitrarily long piece of plaintext a fixed-length bit string. Some of the message digests are MD5 and SHA1 (Secure Hash Algorithm).

To sign a message with a digital signature, a secure hash operation and a private key operation must be performed. Verifying a signature requires a secure hash and a public key operation. The authentication protocol SSL (Secure Sockets Layer) is currently used

for the majority of e-commerce transactions on the World Wide Web. It is used for mutual or one-way authentication and for ensuring the integrity and confidentiality of the data being exchanged. It is a complex transaction that has two phases: a handshake phase and the data transfer phase. In the handshake phase, the client and server use a public-private key algorithm to authenticate each other and exchange a secret. This process requires the server to send the client its public key. The client generates a secret, encrypts it using the server's public key, then sends it to the server. The server decrypts the secret using its private key. At this point, both the client and server have the same secret. They both use this secret to generate a symmetric key for the data transfer operation. During the data transfer phase, the client and server use this symmetric key along with a message digest to communicate securely.

The purpose of this project was to conduct a performance analysis of various security protocols. This includes the study of basic operations of symmetric key and public-private key algorithms, the study of the efficiency of these algorithms with increasing key sizes, and the measurement of the efficiency of secure hash functions. In addition, a comparison study of composite security operations of secure hash functions with symmetric encryption algorithms and secure hash functions with public key algorithms was carried out.

Chapter 2

SECURITY ALGORITHMS, MESSAGE DIGESTS AND SSL

2.1 Symmetric Key Algorithms

Symmetric key algorithms use the same key for encryption and decryption. They use block ciphers, which take an n-bit block of plaintext as input and transform it using the key into an n-bit block of ciphertext. DES and AES are block ciphers. A block cipher is basically a monoalphabetic substitution cipher using big characters, which means whenever the same plaintext block goes in the front end, the same ciphertext block comes out the back end.

2.1.1 DES

DES was widely adopted by the industry for use in security products. With DES, plaintext is encrypted in blocks of 64 bits and yields 64 bits of ciphertext. The algorithm, which is parameterized by a 56-bit key, has 19 distinct stages. [Tanenbaum03]

2.1.2 3DES

According to Tanenbaum, DES key length was too short, so a way was devised to effectively increase it using triple encryption: 3DES, which employs two keys and three

stages. In the first stage, the plain text is encrypted using DES in the usual way with K1. In the second stage, DES is run in decrypted mode, using K2 as the key. Finally, another DES encryption is done with K1. [Tanenbaum03]

2.1.3 AES

AES is based on Rijandael. It is like DES and uses substitution and permutations; it also uses multiple rounds. The number of rounds depends on key size and block size. However, unlike DES, all operations involve entire bytes to allow for efficient implementation in both hardware and software. AES supports three key lengths: 128, 192, and 256 bit block size. [Tanenbaum03]

2.1.4 RC4

RC4 is a stream cipher designed by Ronald Rivest for RSA Data Security. It is a variable key-size stream cipher with byte oriented operations. The algorithm is based on the use of random permutations. The algorithm keystream is completely independent of the plaintext used. RC4 uses a variable length key from 1 to 256 bytes to initialize a 256-byte state table. The state table is used for subsequent generation of a pseudo random stream that is XORed with the plaintext to give the ciphertext. It has the capability of using keys between 1 and 2048 bits. [Vocal03]

2.2 Public-Private Key Algorithms

In public key algorithms, encryption and decryption keys are different. It is not feasible to derive the decryption key could not feasibly be derived from the encryption key. Public key cryptography requires each user to have two keys: a public key, used by the entire world for encrypting messages to be sent to the user, and a private key, which the user needs for decrypting messages.

2.2.1 RSA

The RSA algorithm gets its security from the difficulty of factoring large numbers. Essentially, the public and private keys are functions of a pair of large numbers. Recovering the plaintext from a given ciphertext and the public key used to create it is believed to be equivalent to the problem of recovering the primes used to make the keys. However, it requires keys of at least 1024 bits for good security, which makes it quite slow. [Hook05]

2.2.2 ElGamal

ElGamal is based on the difficulty of solving the discrete logarithm problem. The ElGamal algorithm can be used for both encryption and decryption. The security of the ElGamal scheme relies on the difficulty of computing discrete logarithms over GF(p), where p is a large prime. Prime factorization and discrete logarithms are required to implement the RSA and ElGamal cryptosystems. [Hook05]

2.3 Message Digests

Message digests provide authenticity, but not secrecy. This scheme is based on the idea of a one-way hash function that computes from an arbitrarily long piece of plaintext a fixed-length bit string. Computing a message digest from a piece of plaintext is much faster than encrypting the plaintext with a public key algorithm, so message digests can be used to speed up digital signature algorithms. [Tanenbaum03]

To sign a digital signature, both a secure hash operation and a private key operation need to be performed. Verifying a signature requires a secure hash and a public key operation. Figure 1 depicts the operation of a digital signature.

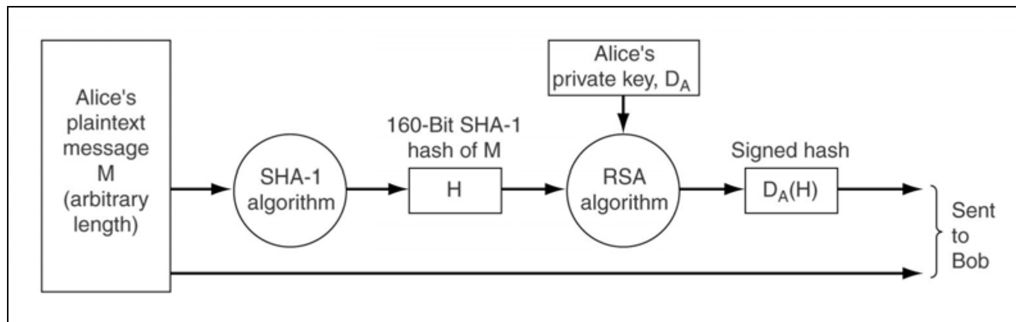

Figure 1: Example of a Digital Signature Scheme [Tanenbaum03]

2.3.1 MD5

MD5 is the fifth in a series of message digests designed by Ronald Rivest. It operates by mangling bits in a sufficiently complicated way so every output bit is affected by every input bit. It starts out by padding the message to a length of 448 bits. Then the original

length of the message is appended as a 64-bit integer to give a total input whose length is a multiple of 512 bits. The last pre-computation step is initializing a 128-bit buffer to a fixed value. [Tanenbaum03]

2.3.2 SHA1

SHA1 was developed by the NSA. SHA1 processes input data in 512-bit blocks, and generates 116-bit message digests. SHA1 pads the message by adding a 1 bit to the end, followed by as many 0 bits as needed to make the length a multiple of 512 bits. Then a 64-bit number containing the message length before padding is ORed into the low-order 64 bits. [Tanenbaum03]

2.4 SSL

SSL builds a secure connection between two sockets including parameter negotiation between client and server, mutual authentication of client and server, secret communication, and data integrity protection. The positioning of SSL in the usual protocol stack is illustrated in Figure 2. It is a new layer interposed between the application layer and the transport layer, accepting requests from the browser and sending them down to TCP for transmission to the server. [Tanenbaum03]

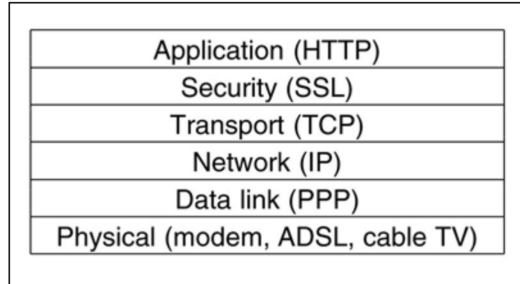

Figure 2: Layers for User Browsing With SSL [Tanenbaum03]

Once a secure connection has been established, SSL's main job is handling compression and encryption. Figure 3 shows a simplified version of the SSL connection establishment protocol.

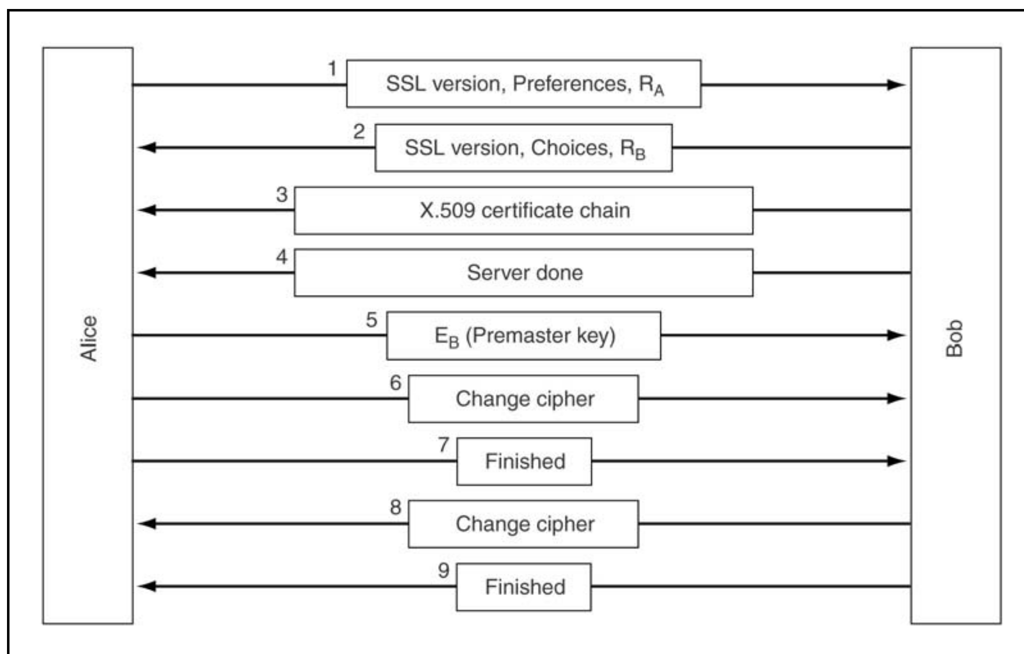

Figure 3: Simplified Version of the SSL Connection Establishment Protocol [Tanenbaum03]

SSL supports multiple cryptographic algorithms. 3DES with SHA1 and RC4 with MD5 are commonly used combinations. 3DES and RC4 are used for encryption and SHA1 and MD5 are used for message integrity.

Chapter 3

PROJECT DESCRIPTION

3.1 Overview

The overall goal of the study was to measure the performance of various security algorithms. The project implemented a client and server application, enabling the client and server to send encrypted messages to each other. As an example, the server encrypted the message, measured the time it took to encrypt the message, and sent the message to the client. The client decrypted the message it received and measured the decryption time.

The project was divided into two parts. First, we studied the performance of the basic operations of symmetric key encryption algorithms (DES, 3DES, and AES) and public-private key algorithms (RSA, ElGamal), with increasing key size for operations using the open source Bouncy Castle library and the efficiency of secure hash functions (MD5 and SHA1). Second, we compared composite security operations. This included the performance analysis of digital signatures, specifically RSA with MD5 and RSA with SHA1. Also, included is the comparison study of the SSL protocols of RC4 with MD5 and 3DES with SHA1, using open SSL software.

The first part of the project was as follows:

- Performed basic operations and compared DES versus RC4 with 64-bit key size and with various file sizes from 1K to 10K.
- Performed basic operations and compared various symmetric algorithms, specifically, 3DES versus RC4 versus AES with key size of 192 bits with various file sizes from 1K to 10K.
- Compared the encryption and decryption times of RC4 versus AES with a 5K file with varying key sizes from 64 bits to 2048 bits.
- Compared the encryption and decryption times of RC4 versus AES for a 10K file and varying key sizes from 64 bits to 2048 bits.
- Compared the encryption times of the RSA algorithm for various files sizes (100 bytes, 1K, 5K, 10K) with different key sizes (1024 bits, 2048 bits, 3072 bits, 4096 bits, 5120 bits).
- Compared the encryption times of ElGamal algorithm for different files sizes (100 bytes, 1K, 5K, 10K) with different key sizes (600 bits, 800 bits, 1000 bits, 1200 bits).

The following outlines the secod part of the study:

- Compared the digital signature verification times of the RSA algorithm with the SHA1 hash function versus the RSA algorithm with the MD5 hash function for a 1MB file with different key sizes (512 bits, 768 bits, 1024 bits, 2048 bits).
- Determined the maximum throughput for digital signatures for the RSA algorithm with the SHA1 hash function and the RSA algorithm with the MD5 hash function for a 1MB file with different key sizes (512 bits, 768 bits, 1024 bits, 2048 bits).

- Measured the maximum throughput of two different SSL setups: one used RC4 and MD5 and the another used 3DES and SHA1 with different RSA key sizes (512 bits, 768 bits, 1024 bits, 2048 bits).

3.2 Hardware

The hardware for this project consisted of two Intel based workstations with the Microsoft XP operating system, version 2002, service pack 2. Each workstation had an Intel Pentium 3 GHZ processor and 1 GB RAM. They were connected by 100 megabit Ethernet.

3.3 Software

The software for this project consisted of a Java 2 Runtime environment, standard edition, version 1.4. We used Java for developing the security algorithms and open source Bouncy Castle software Release 1.32 for implementing the algorithms. The software for implementing the SSL project consisted of Cygwin, version 1.5.21-1. We used open source Open SSL Library, version 0.9.8, for implementing the SSL security algorithms. C was used to develop the SSL applications.

Chapter 4

TESTING METHODOLOGY

4.1 Security Algorithms Testing Method

We ran client and server applications on two different workstations. The client connected to the server at a specific port number and provided a list of security algorithms with different key lengths. Upon selecting a security algorithm, the client was prompted for a test file with encrypted plaintext, which was then sent to the server. Upon receiving the client test file, the server decrypted the received text and saved it to a file. We recorded the encrypting and decrypting timings at the client and server with increasing file sizes and increasing key sizes.

4.2 SSL Testing Method

We ran the security server and the security client on two different workstations. The client began by connecting to the server and providing authentification to the server by handshaking. After successfully handshaking, the client provided a list of security algorithms with different RSA algorithm key lengths. Upon selecting a particular security algorithm, the client was prompted for a test file and sent encrypted text to the server. Upon receiving the client test file, the server decrypted the received text and saved it to a file. We recorded the encrypting and decrypting timings for the client and server.

Chapter 5

RESULTS

5.1 DES Versus RC4 Encryption and Decryption Times

We ran the client and server programs for the DES and RC4 security algorithms with a 64 bit key for different test file sizes. Test file sizes ranged from 1K to 10K, varying in size by 1K increments . We measured encryption and decryption times at the client and server and graphs were plotted representing the measured times.

In Figure 4 we can see the DES algorithm encryption and decryption times increase almost linearly as the file size increases. DES encryption times increased by a factor of 1.017 to 1.09 with each 1K increment in the test file size. The DES decryption times increased by a factor of 1.03 to 1.1 times with each 1K increment in the test file size. Like all symmetric key algorithms, the DES decryption times are 1.2 to 1.3 times faster than encryption times.

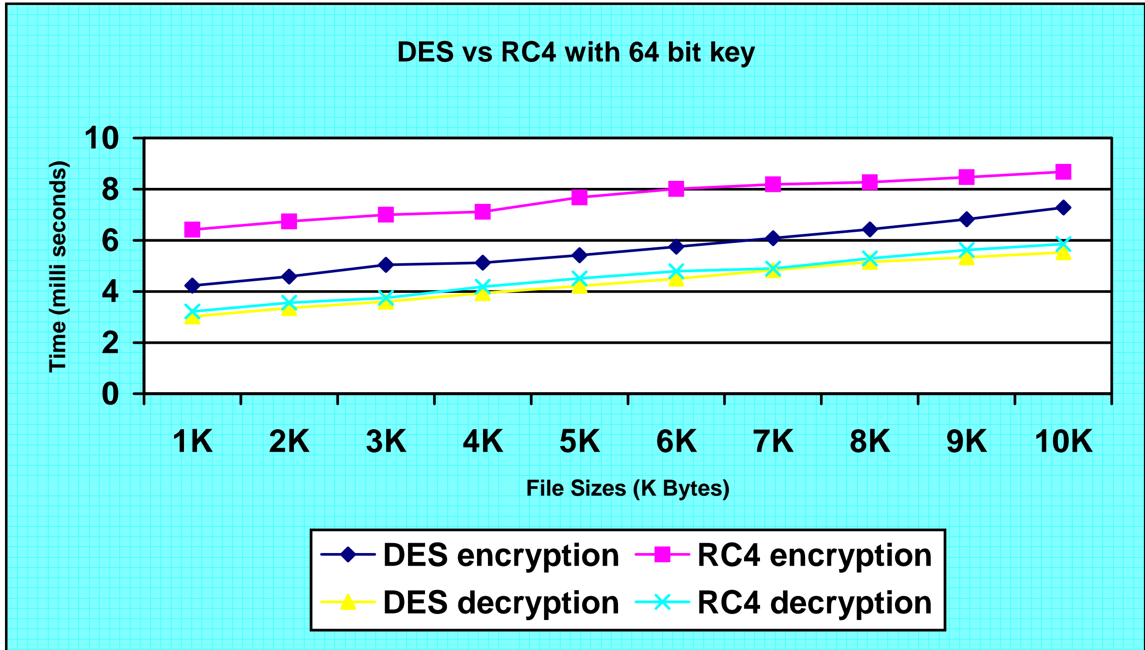

Figure 4: DES versus RC4 with 64 Bit Key Encryption and Decryption

In the above the figure we can also see that the RC4 encryption and decryption times linearly increase as the file size increases. The RC4 encryption times increase by a factor of 1.009 to 1.04 for each 1K increment in test file size. The RC4 decryption times increase by a factor of 1.02 to 1.1 times with each 1K increment in test file size. Like all symmetric key algorithms, the RC4 decryption times are 1.4 to 1.99 times faster than encryption.

Figure 4 shows the DES security algorithm had better response times than the RC4 algorithm. DES encryption times are 1.19 to 1.5 times faster than those of RC4 encryption times. DES decryption times are 1.014 to 1.069 times faster than those of RC4 decryption times.

5.2 3DES Versus RC4 Versus AES Encryption and Decryption Times With 192 Bit Key

We ran the client and server programs for the 3DES, RC4, and AES security algorithms with a 192 bit key. Test file sizes ranged from 1K to 10K, varying in size by 1K increments. We measured encryption and decryption times at the client and server, and graphs were plotted representing the measured times.

In Figures 5 and 6 we can see 3DES encryption rates increased by a factor of 1.006 to 1.043 for each 1K increment in test file size and decryption rates increased by a factor of 1.042 to 1.097 for each 1K increment in test file size. 3DES decryption times are 1.18 to 1.53 times faster than encryption times.

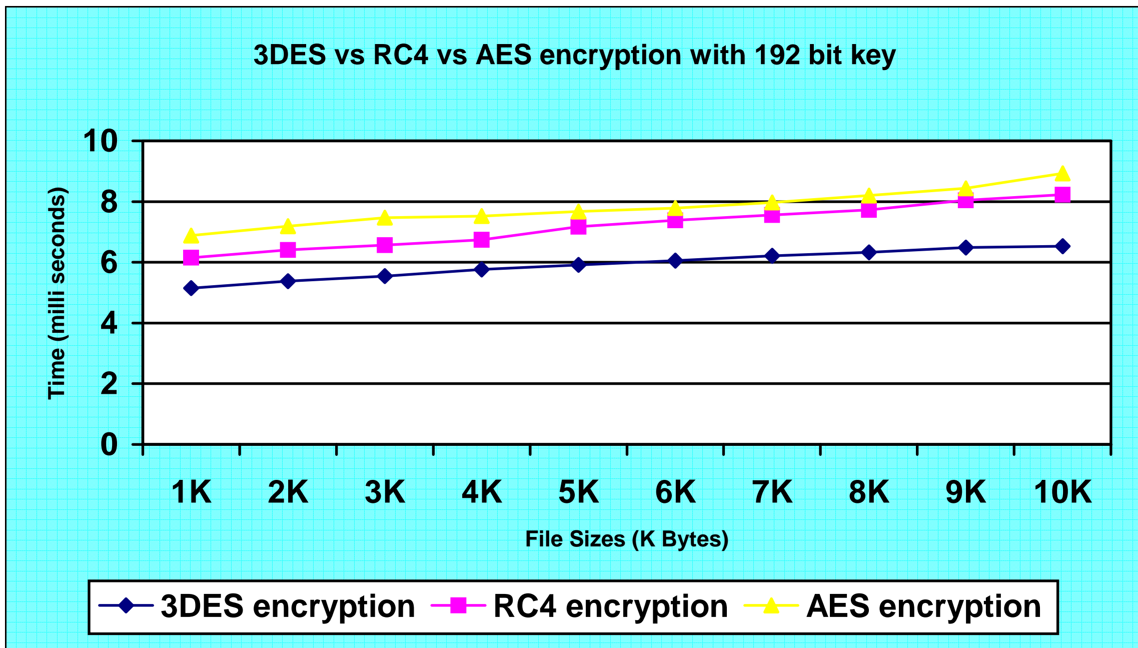

Figure 5: 3DES versus RC4 versus AES Encryption with 192 Bit Key

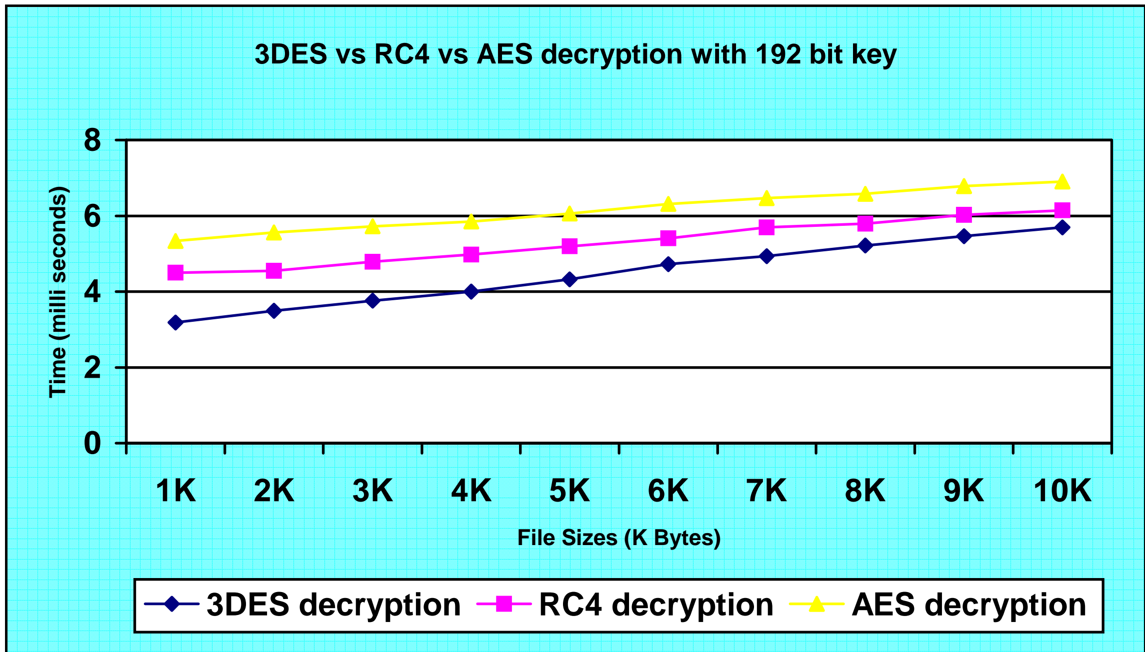

Figure 6: 3DES versus RC4 versus AES Decryption with 192 Bit Key

From Figures 5 and 6 we can see RC4 encryption times increased by a factor of 1.02 to 1.06 for each 1K increment in test file size. RC4 decryption times increased by a factor of 1.01 to 1.05 for each 1K increment in test file size. RC4 decryption rates are 1.3 to 1.4 times faster than encryption rates.

In Figures 5 and 6 we can see AES encryption times increased by a factor of 1.02 to 1.057 for each 1K increment in test file size. AES decryption times increased by a factor of 1.02 to 1.04 for each 1K increment in test file size. AES decryption times are 1.2 to 1.3 times faster than encryption times.

From Figures 5 and 6 it can be seen that the 3DES security algorithm had better response times than the RC4 and AES algorithms and RC4 performed better than AES. 3DES encryption times are 1.16 to 1.24 times faster than RC4 encryption times and 1.29 to 1.36

times faster than AES encryption times. 3DES decryption times are 1.07 to 1.4 times are faster than RC4 decryption times and 1.29 to 1.36 times faster than AES decryption times. RC4 encryption times are 1.28 to 1.36 times faster than AES encryption times and RC4 decryption times are 1.21 to 1.67 times faster than AES decryption times.

5.3 RC4 Versus AES With 5K File

We ran the client and server programs for the RC4 and AES security algorithms with a 5K file for various key sizes. Key sizes included 128 bits, 192 bits, and 256 bits. We measured encryption and decryption times at the client and server and graphs were plotted representing the measured times.

Figure 7 shows RC4 and AES encryption times for a 5K file and Figure 8 shows RC4 and AES decryption times for a 5K file. RC4 encryption times increased by a factor of 1.019 and 1.016 for 192 and 256 bit keys respectively. RC4 decryption times are increased by a factor of 1.08 and 1.04 for 192 and 256 bit keys respectively. RC4 decryption times are 1.51 to 1.64 times faster than RC4 encryption times.

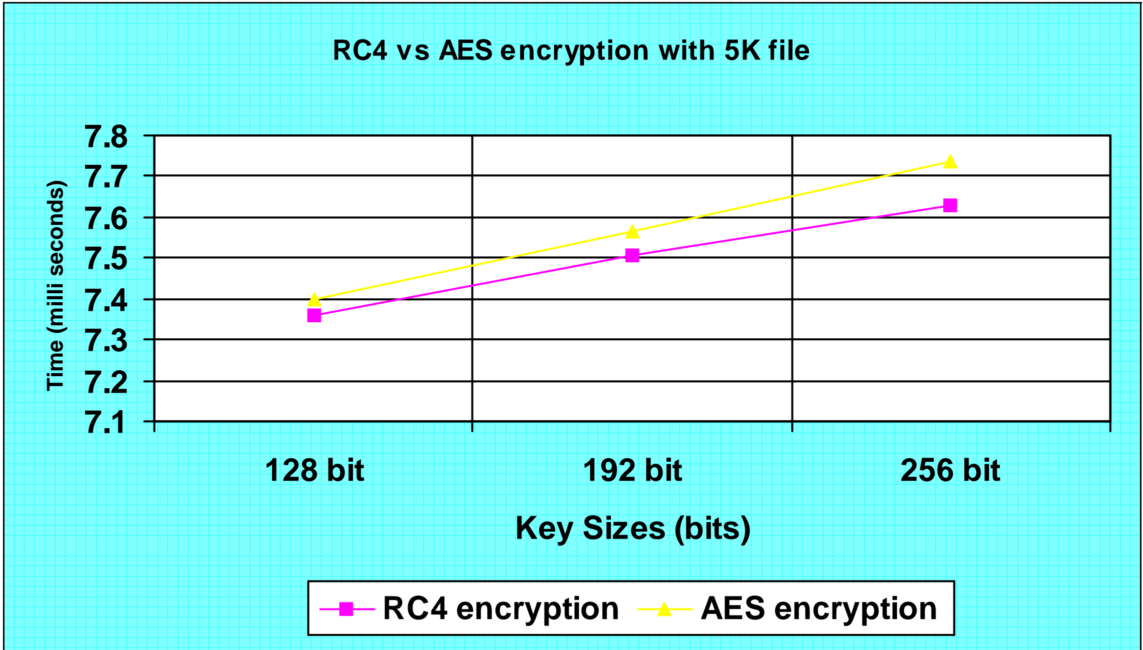

Figure 7: RC4 versus AES Encryption Times With 5K File

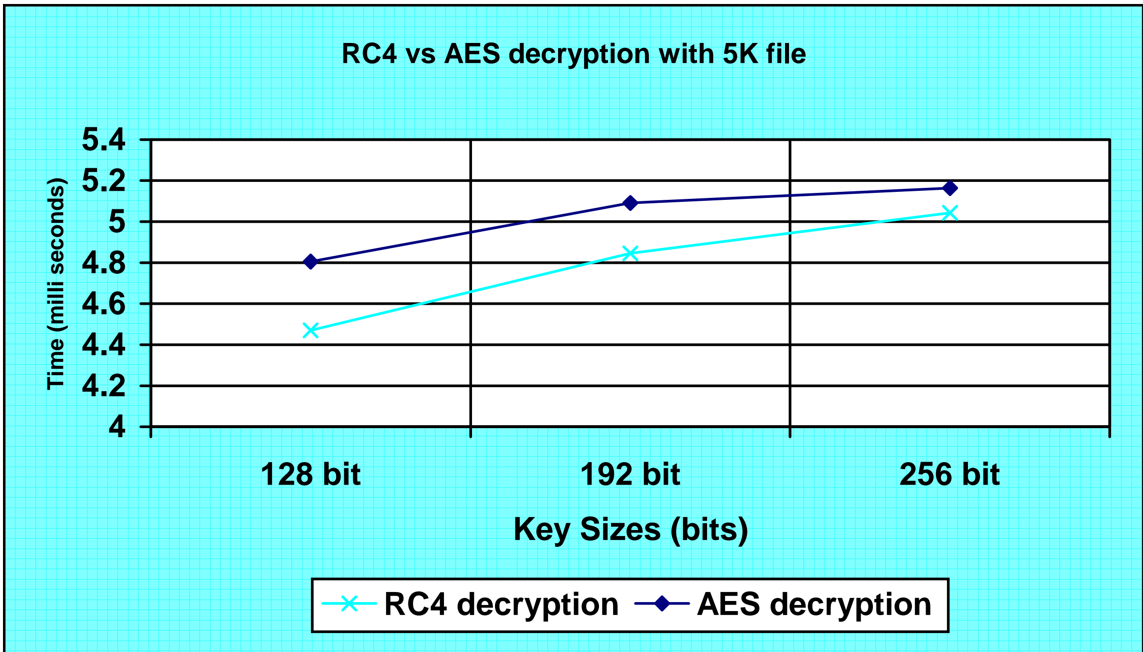

Figure 8: RC4 versus AES Decryption With 5K File

In Figures 7 and 8, we also can see AES encryption times increasing by a factor of 1.019 and 1.016 for 192 and 256 bit keys respectively. AES decryption times increase by a factor of 1.059 and 1.014 for 192 and 256 bit keys respectively. AES decryption times are 1.48 to 1.53 times faster than AES encryption times.

In Figures 7 and 8, we observed that the RC4 security algorithm had better response times than the AES algorithm. RC4 encryption times are 1.005 to 1.014 times faster than AES encryption times. RC4 decryption times are 1.024 to 1.05 times faster than AES decryption times.

5.4 RC4 Versus AES With 10K File

We ran the client and server programs for the RC4 and AES security algorithms with a 10K file for various key sizes. Key sizes included 128 bits, 192 bits, and 256 bits. We measured encryption and decryption times at the client and server, and graphs were plotted representing the measured times.

Figures 9 and 10 show RC4 and AES encryption and decryption times for a 10K file. RC4 encryption times increased by a factor of 1.0197 and 1.037 for 192 and 256 bit keys. RC4 decryption times increased by a factor of 1.043 and 1.037 for 192 and 256 bit keys respectively.

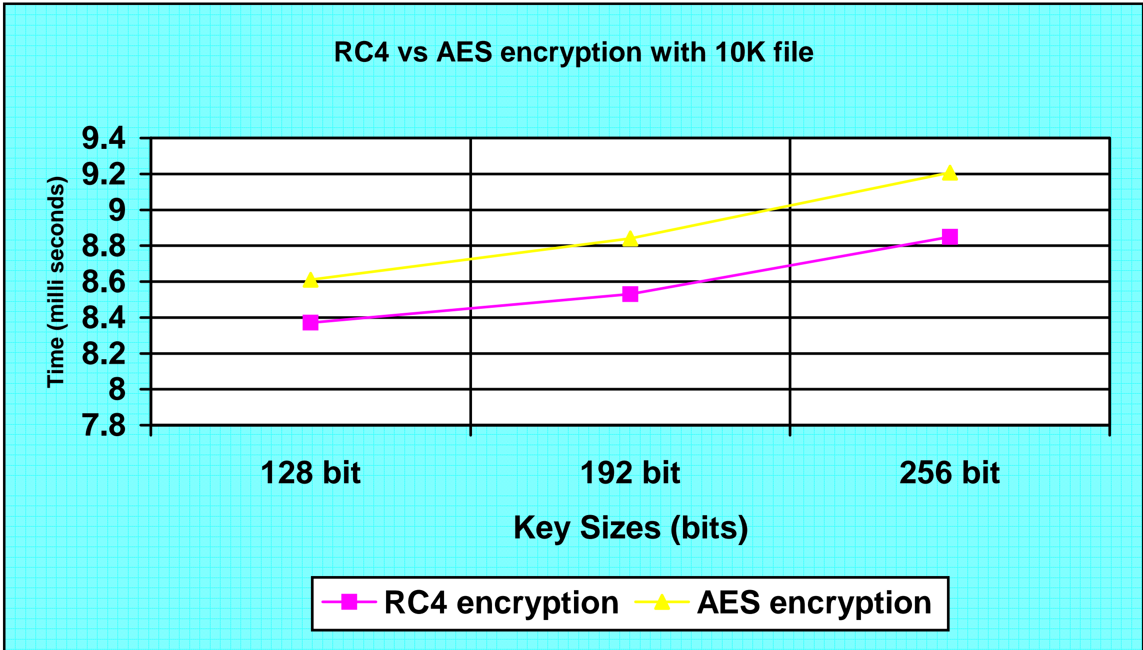

Figure 9: RC4 versus AES Encryption With 10K File

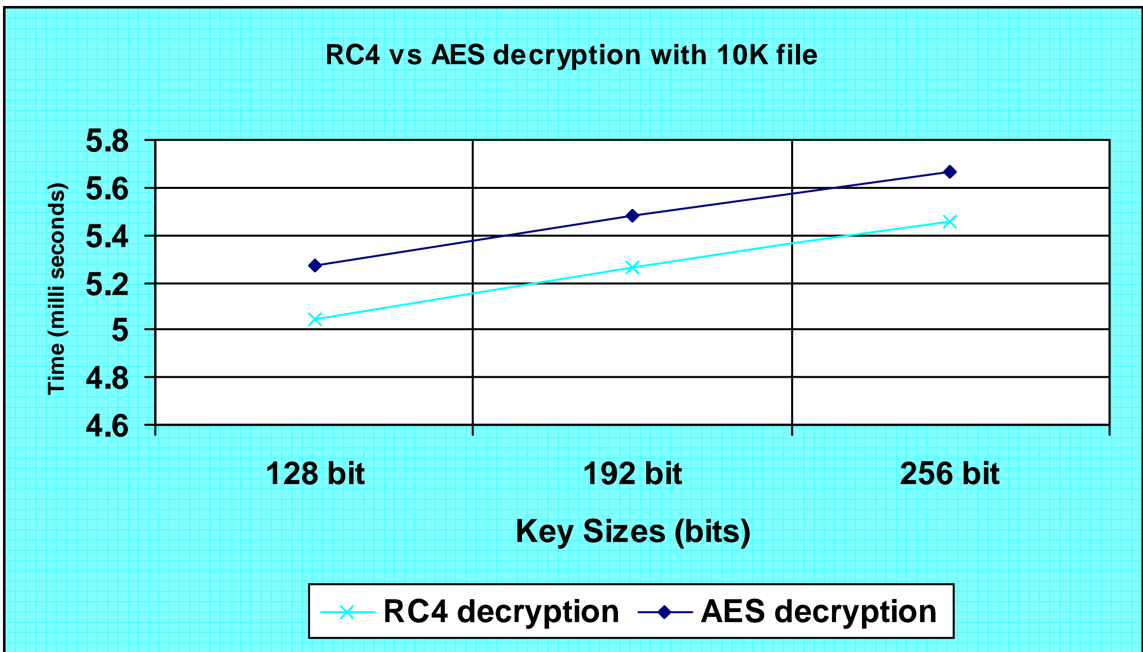

Figure 10: RC4 versus AES Decryption With 10K File

AES encryption times increased by a factor of 1.02 and 1.04 for 192 and 256 bit keys. AES decryption times increased by a factor of 1.04 and 1.03 for 192 and 256 bit keys.

As seen in both figures, the RC4 security algorithm had better response times than the AES algorithm. RC4 algorithm encryption times are 1.028 to 1.04 times faster than AES encryption times. RC4 algorithm decryption times are 1.03 to 1.044 times faster than those produced by the AES algorithm.

5.5 RSA Algorithm Encryption and Decryption

We ran the client and server programs for the RSA security algorithm employing various test files and key sizes. Test file sizes included 100 bytes, 1K, 5K, and 10K. Key sizes included 1024 bits, 2048 bits, 3072 bits, 4096 bits, and 5120 bits. We measured encryption and decryption times on the client and server and graphs were plotted representing measured times.

In Figure 11 we see, for every 1024 bits increment in key size, public key encryption times increased by a factor of 1.16 to 1.26 for the 100 byte file. In Figure 12 we see, encryption times increased by a factor of 1.32 to 2.72 for the 1K file, 1.53 to 2.73 for the 5K file, and 1.56 to 2.58 for the 10K file.

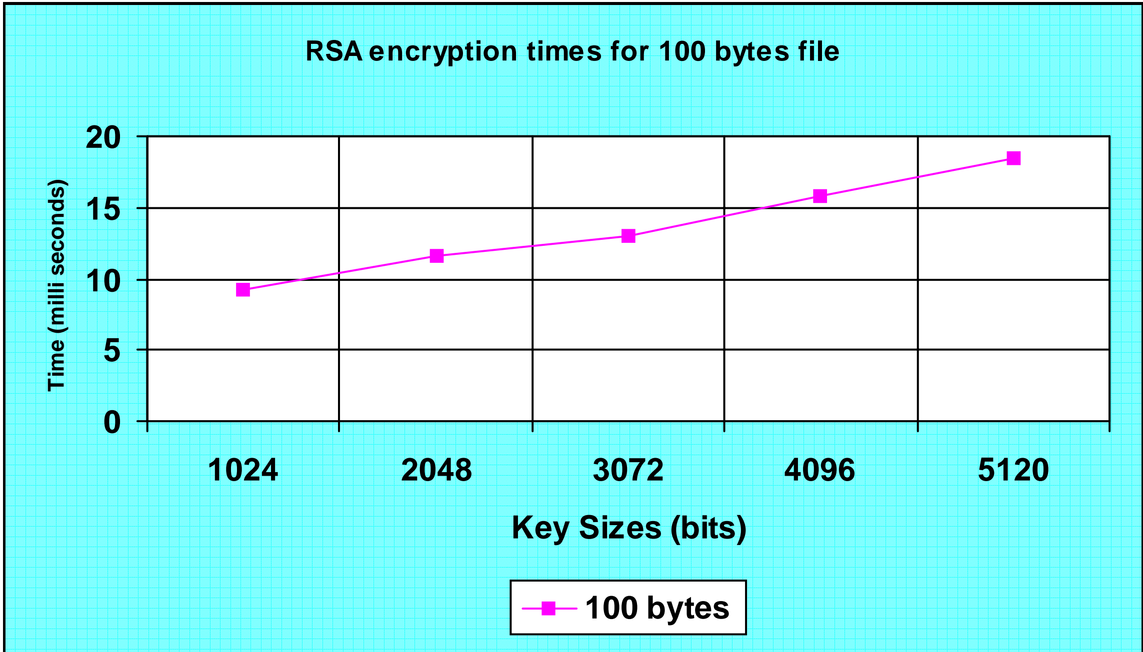

Figure 11: RSA Encryption Times for 100 bytes file

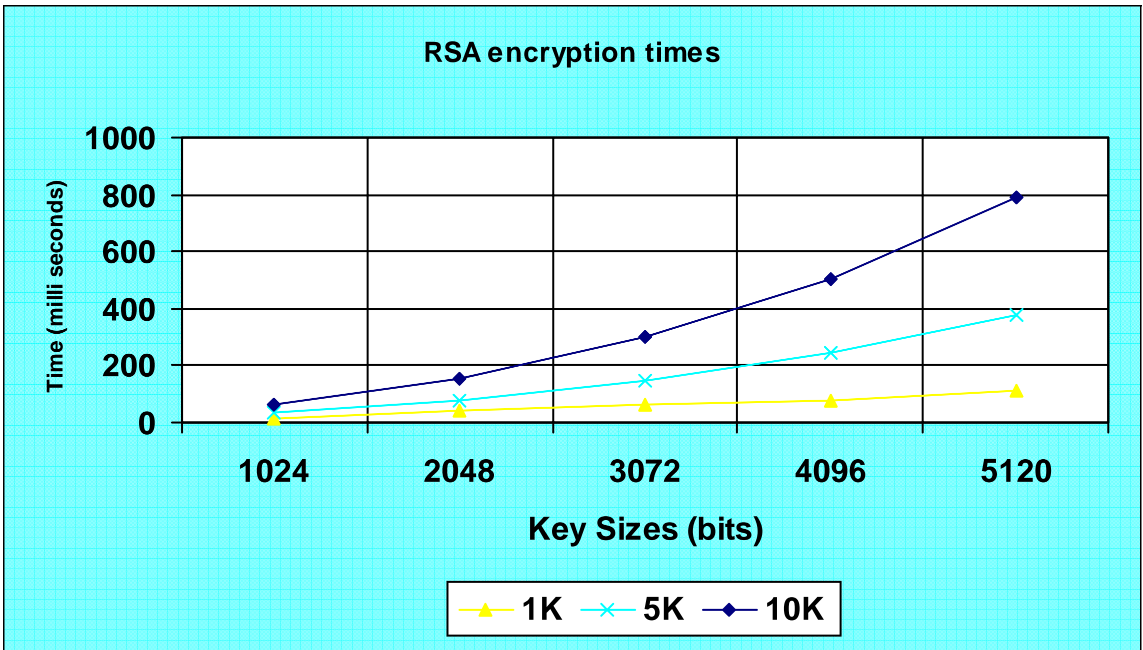

Figure 12: RSA Encryption Times

In Figure 13 we see, for every increment of 1024 bits in key size, decryption times increased by a factor of 1.94 to 5.98 for the 100 byte file. In Figure 14 we see, the

corresponding increase in decryption times are 1.9 to 6.4 for the 1K file, 1.9 to 6.8 for the 5K file, and 1.9 to 7.06 for the 10K file.

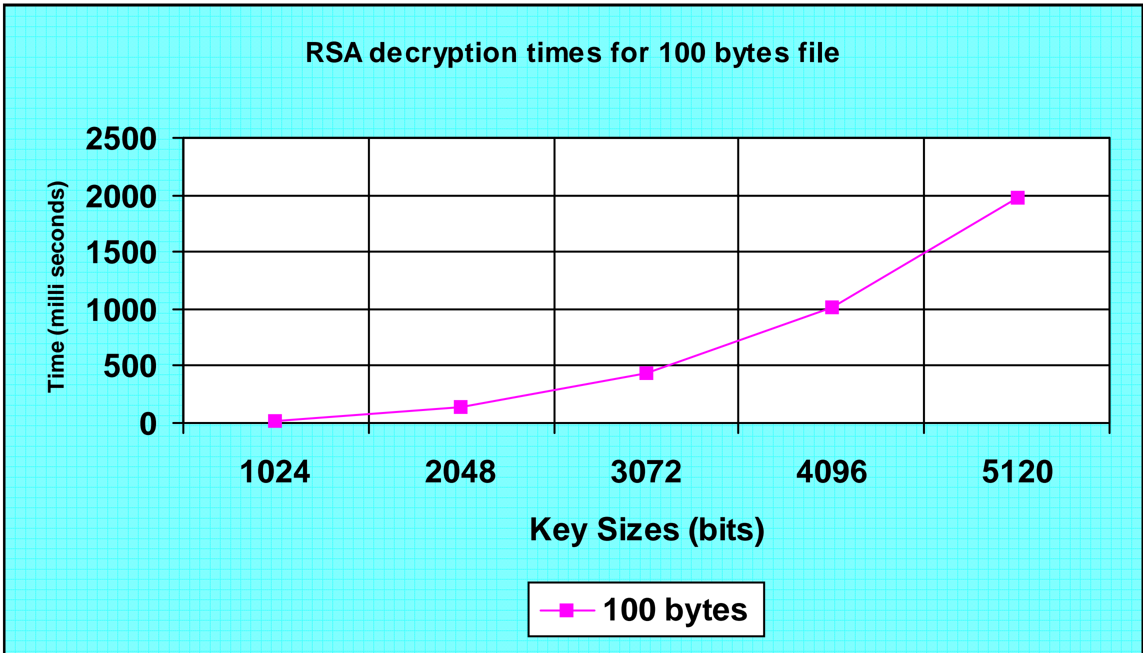

Figure 13: RSA Decryption Times for 100 bytes file

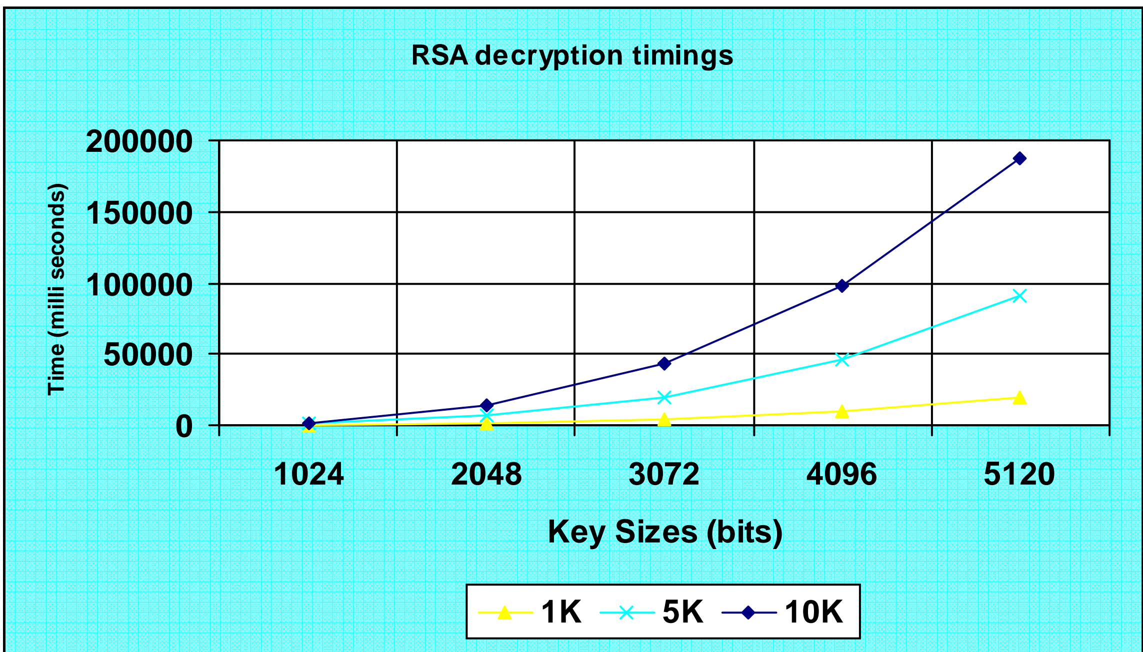

Figure 14: RSA Decryption Times

The figures also show that RSA algorithm decryption times were slower than encryption times. The ratio between private and public key operation times grows linearly with the key length. For the file of size 100 bytes, the ratio between private and public key operation times increased from 2.6 to 107 times with key size increase. For a 1K file this ratio increased from 13 to 175 times with key size increase; for the 5K file, this ratio increased from 29 times to 240; and for 10K file this ratio increased from 33 to 236 times. So the file size and key size impact RSA performance. This indicates performance is a tradeoff for security.

5.6 ElGamal Encryption and Decryption Times

We ran the client and server programs for the ElGamal security algorithm for various test file and key sizes. Test file sizes included 100 bytes, 1K, 5K, and 10K. Key sizes included 600 bits, 800 bits, 1000 bits, and 1200 bits. We measured encryption and decryption times on the client and server and graphs were plotted representing the measured times.

As seen in Figures 15 and 16, with increasing key size, public key encryption times increased by a factor of 1.49 to 2.25 for the 100 byte test file, 1.63 to 2.1 for the 1K test file, 1.66 to 2.2 for the 5K test file, and 1.62 to 2.21 for the 10K test file. Figures 17 and 18 shows, with increasing key size, ElGamal decryption times increased by a factor of 1.65 to 1.85 for the 100 byte test file, 1.6 to 2.1 for 1K test file, 1.5 to 2.2 for 5K test file, and 1.65 to 2.3 for the 10K test file.

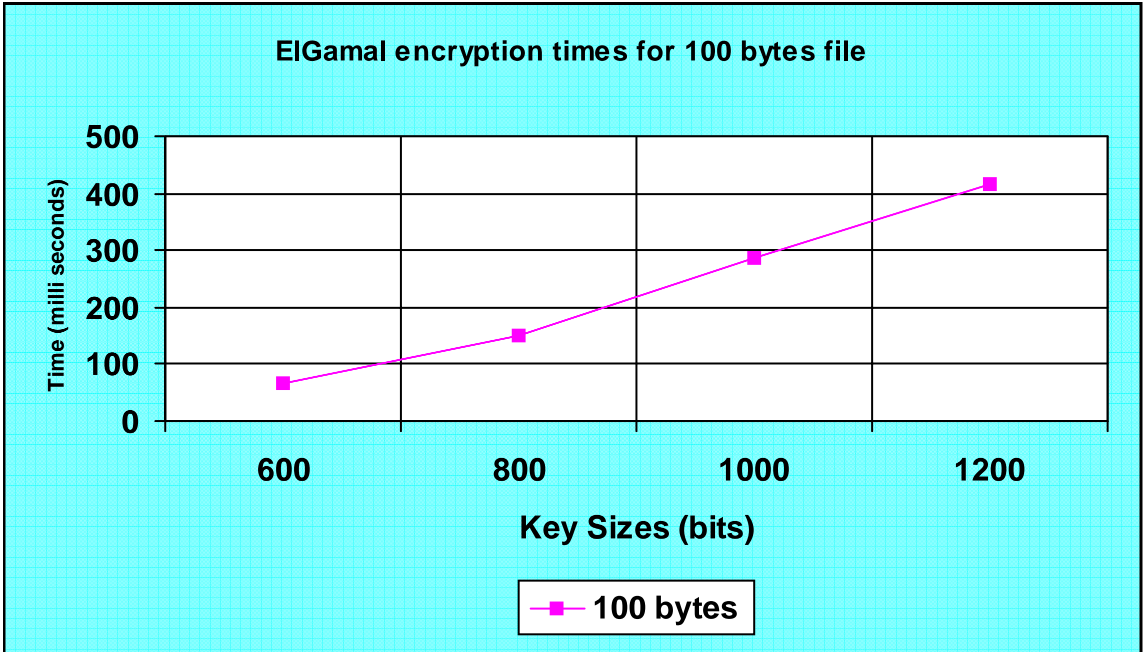

Figure 15: ElGamal Encryption Times for 100 bytes file

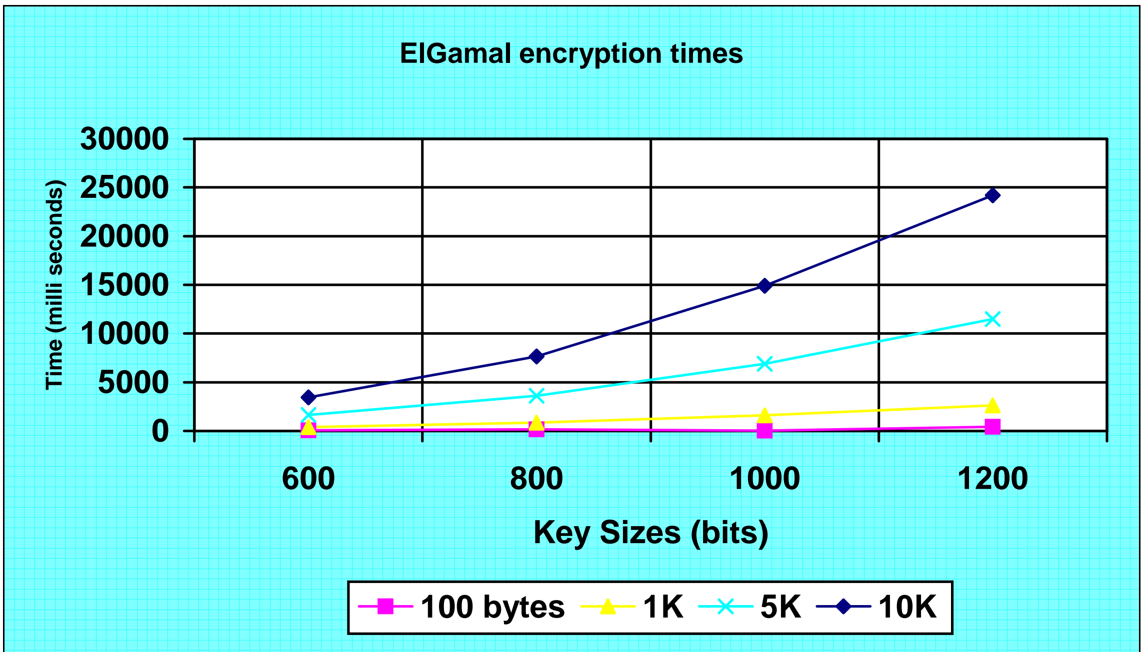

Figure 16: ElGamal Encryption Times

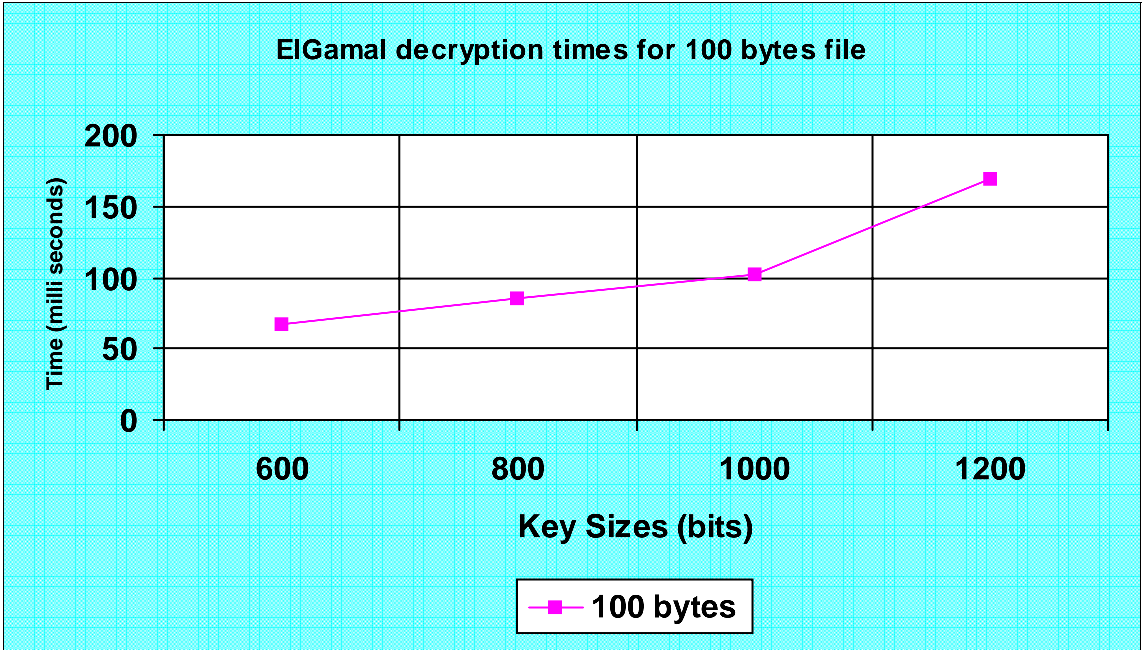

Figure 17: ElGamal Decryption Times for 100 bytes file

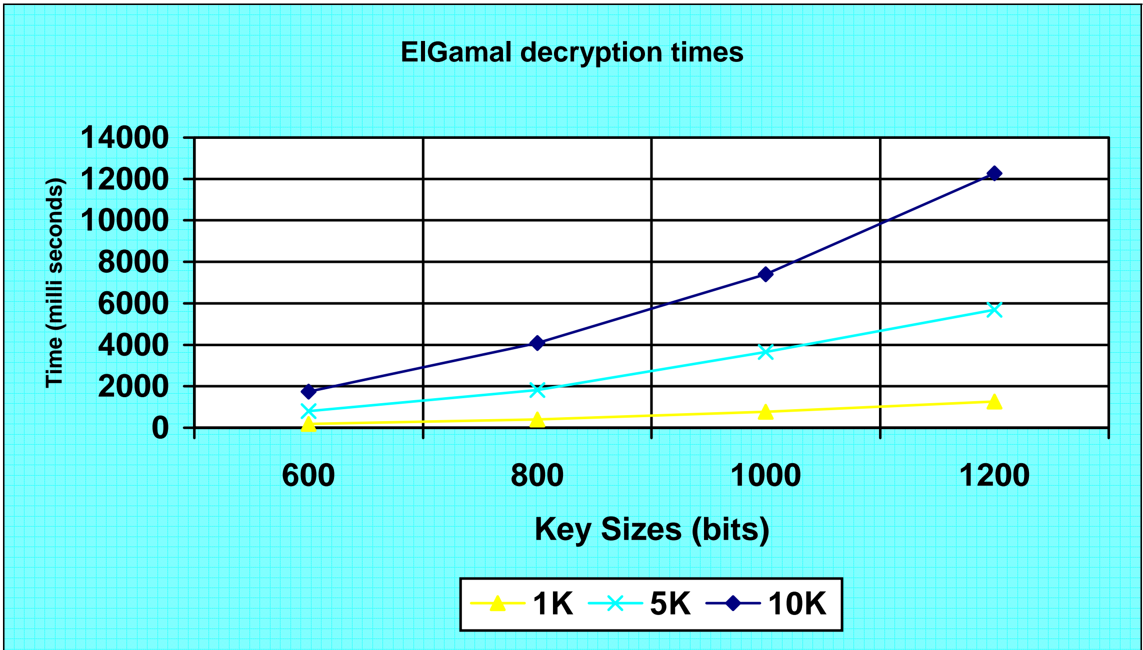

Figure 18: ElGamal Decryption Times

For the ElGamal algorithm, decryption times were faster than encryption times. Private key decryption times are about two times faster than public key encryption times for all

test files. In contrast to the RSA algorithm, the ElGamal algorithm public key operation is slower than the private key operation.

5.7 Digital Signature SHA1 Versus MD5 With RSA

We ran the client and server programs for digital signature with SHA1 and MD5 with various RSA key sizes. Key sizes included 512 bits, 768 bits, 1024 bits, and 2048 bits. A 1MB test file was used. We measured digital signature verification times on the client and server and graphed the measured times.

Figure 19 shows the digital signature verification times with SHA1 increased by a factor of 1.09 to 1.23 for each increment of RSA key size. Digital signature verification times with MD5 increased by a factor of 1.01 to 1.3 for each increment of RSA key size.

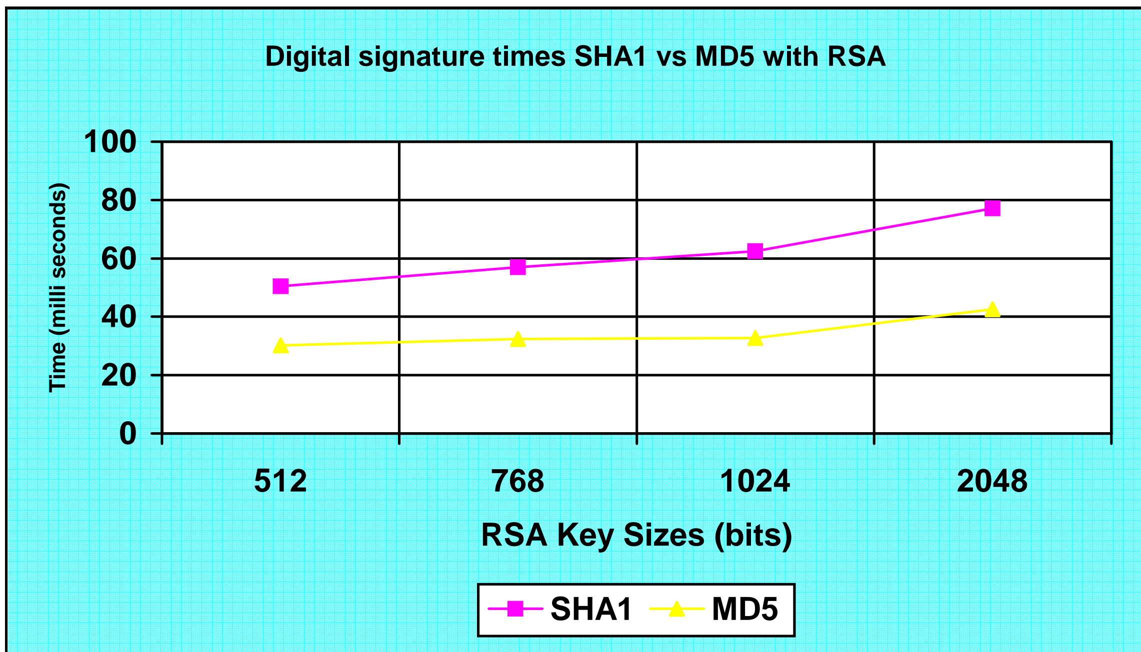

Figure 19: Digital Signature Verification Times SHA1 versus MD5 with RSA

Figure 20 shows digital signature maximum throughput for SHA1 and MD5. As RSA key size increses, SHA1 throughput decrements at the rate of 0.88, 0.91, and 0.80 while the MD5 throughput decrements at the rate of 0.94, 0.91, and 0.74.

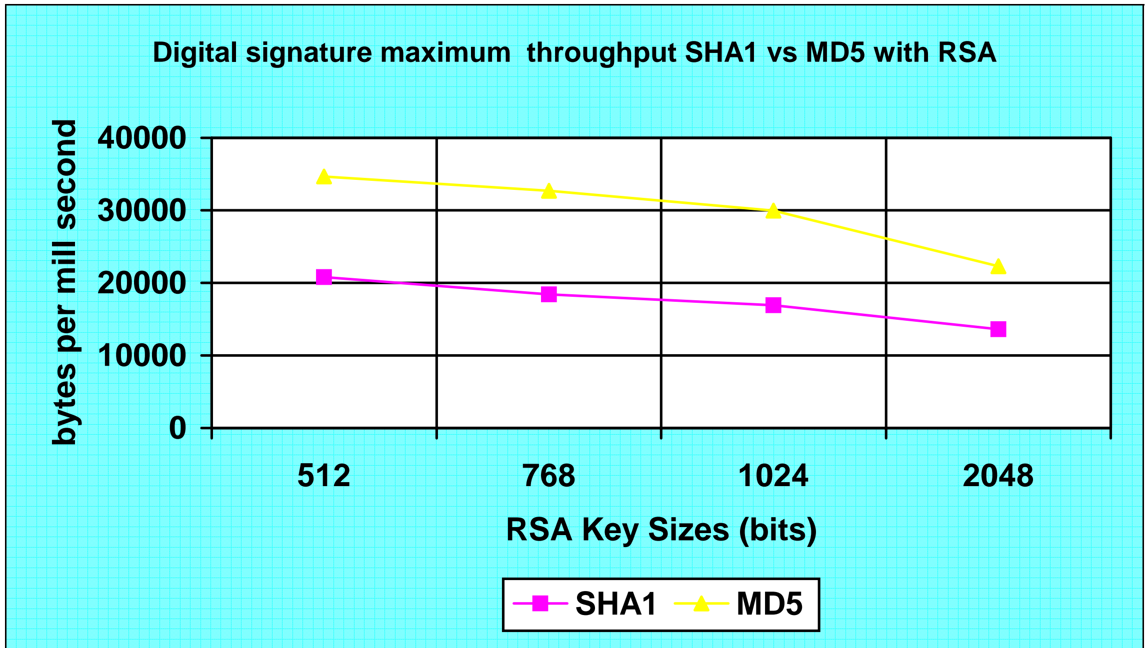

Figure 20: Digital Signature Maximum Throughput SHA1 versus MD5 with RSA

Figure 20 illustrates our finding that digital signatures with MD5 exhibit better throughput than digital signatures with SHA1. MD5 digital signature rates are faster by 1.61 to 1.77 times for RSA key sizes 512, 768, 1024, and 2048 bits.

5.8 SSL Communication Timings for Two Sets of Security Algorithms

We ran the client and server programs for SSL communication with the 3DES-SHA1 and RC4-MD5 security algorithms with various RSA key sizes. RSA key sizes included 512

bits, 768 bits, 1024 bits, and 2048 bits. A 10K test file was used. We measured encryption and decryption times and graphed the measured times.

In Figure 21 we can see, as the key size increased, 3DES-SHA1 throughput rates decreased at the rates of 0.61, 0.85, 0.91 and RC4-MD5 throughput rates decreased at the rate of 0.66, 0.86, and 0.87. We also observed that RC4-MD5 had better throughput than the 3DES-SHA1 combination. RC4-MD5 was 1.052 to 1.09 times faster than 3DES-SHA1.

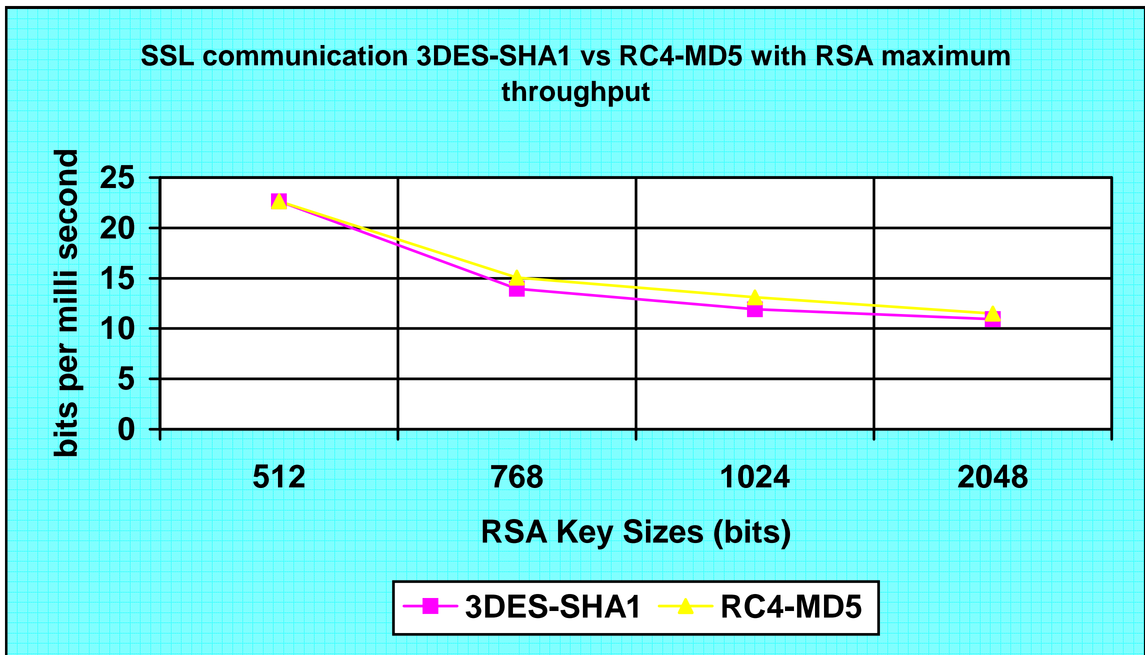

Figure 21: SSL Communication 3DES-SHA1 versus RC4-MD5 with RSA Maximum Throughput

Chapter 6

ANALYSIS AND CONCLUSIONS

6.1 Analysis of Test Results

Even if a cipher is unbreakable by exploiting structural weaknesses in the algorithm, it is possible to run through the entire space of keys in what is known as a brute force attack. Since longer keys require more work for a brute force search, a long enough key will require more work than is feasible. Thus, length of the key is important in resisting this type of attack [Wikipedia01]. Symmetric algorithms are more susceptible to brute force attack.

When deciding upon key length in the design or use of a cryptographic algorithm, one must consider two tradeoffs. Long keys can provide more security. Short keys can provide greater efficiency. Therefore, it is important to determine an optimal key length by evaluating the likelihood of the key being guessed, versus the impact of a longer key on the time required to encrypt the plaintext and the time required by the intended receiver to decrypt the ciphertext [Williams02]. Out test results show symmetric algorithm encryption and decryption times increase with key size. This proved to be true in our tests of DES, 3DES, RC4, and AES.

The effectiveness of public key cryptosystems depends on the intractability (computational and theoretical) of certain mathematical problems such as integer

factorization. These problems are time consuming to solve, but usually faster than trying all possible keys by brute force. Thus, asymmetric algorithm keys must be longer for equivalent resistance to attack than symmetric algorithm keys. As of 2002, a key length of 1024 bits was generally considered the minimum necessary for the RSA encryption algorithm[Wikipedia01].

Private key operations are generally slower than those with public keys. The ratio between private and public key operation times grows almost linearly with key length and longer keys provide an increased level of security at a performance cost [Manasce03]. Our test results confirm this and show the RSA algorithm private key based decryption times were slower than public key based encryption operations.

Encryption under ElGamal requires two exponentiations; however, these exponentiations are independent of the message and can be computed ahead of time if need be. The ciphertext is twice as long as the plaintext, which is a disadvantage as compared to some other algorithms. Decryption only requires one exponentiation [Wikipedia02]. This explains why our ElGamal public key encryption times were slower than the private key decryption times. This is in contrast to our results with the RSA algorithm in which private key decryption times were slower than pubic key encryption operations.

Verification time increases as security increases, but maximum throughput decreases with security [Menasce03]. We examined the performance of SHA1 and MD5 with the RSA algorithm. Digital certification verification times increased with an increase in RSA

key size. Digital certificate verification with MD5 was faster and had better throughput than with SHA1. These results suggest SHA-1 is more secure than MD5 and the longer the key, the stronger the security.

6.2 Security Algorithms Performance Comparison From Previous Studies

We encountered few previous performance studies on security algorithms. The following sections compare these studies to our project results.

6.2.1 Crypro++ 5.2.1 Benchmarks

Crypto++ 5.2.1 are speed benchmarks for some of the most commonly used cryptographic algorithms. All were coded in C++, compiled with Microsoft Visual C++ .NET 2003 (whole program optimization, optimized for speed, P4 code generation), and run on a Pentium 4 2.1 GHz processor under Windows XP SP 1. For multiple-precision addition and subtraction, 386 assembly routines were used[Wei04].

Table 1 shows the throughput (mb/seconds) for various security algorithms using Crypto++ software. The DES algorithm has better throughput than the RC2 and RC6 algorithms. AES with 192 bit key has better throughput than 3DES algorithm. These results are consistent with the results of our study, in which DES encryption times were faster than RC4 encryption times, suggesting better throughput. AES has better throughput than the DES-EDE3 algorithm, which conflicts the results of our study. Our

AES verification times were slower than those produced when the 3DES algorithm was used so, in our study 3DES had better throughput than AES.

Algorithm	Megabytes(2^20 bytes) Processed	Time Taken	MB/Second
MD5	1.02e+003	4.726	216.674
SHA-1	256	3.766	67.977
SHA-256	256	5.758	44.460
SHA-512	64	5.618	11.392
DES	128	5.998	21.340
DES-EDE3	64	6.499	9.848
RC2	64	5.548	11.536
RC6	128	3.385	37.814
AES (Rijndael) (128-bit key)	256	4.196	61.010
AES (Rijindael) (192 bit key)	256	4.817	53.415
AES (Rijindael) (256 bit key)	256	5.308	48.229

Table 1: Throughput for Security Algorithms Using Crypto++ [Wei04]

Table 2 shows the performance of the RSA algorithm using Crypto++ software. Private key decryption operations are in general slower than public key encryption operations. Verification times increase as security increases. Our project results are consistent with the results of the above study. We tested the performance of public key algorithms, RSA and ElGamal, with various key sizes. Both RSA and ElGamal encryption and decryption times increased with an increase in key sizes. RSA private key based decryption times were slower than public key based encryption operations. ElGamal public key encryption times were slower than private key decryption operations.

Operation	Iterations	Total Time	Milliseconds/Operation
RSA 1024 Encryption	27607	5.007	0.18
RSA 1024 Decryption	1050	5.007	4.77
RSA 2048 Encryption	11022	5.007	0.45
RSA 2048 Decryption	177	5.028	28.41

Table 2: Performance of RSA Public Key Algorithm Using Crypto++ [Wei04]

6.2.2 Testing a Variety of Encryption Technologies

Henson reviewed and tested the speeds of various encryption technologies using Entrust Software [Henson01]. Multiple encryption algorithms were included in the product. The algorithms tested were IDEA, CAST, DES, and RC2. For testing they used a 7.7MB Word document file that included complex graphics and timed encryption, decryption, and signing [Henson01].

Table 3 shows the encryption and decryption times for various security algorithms using Entrust software. All of the symmetric algorithm encryption times are slower than the decryption times. The DES algorithm is faster than 3DES, RC2-40, and RC2-120, since it uses a 64 bit key. 3DES, which uses a 192 bit key, exhibited the same performance as RC2 with a 40 bit key and RC2 with a 128 bit key.

Algorithm	Encrypt Time	Decrypt Time	Sign Time	Decrypt Signing Time	Encrypt & Sign Time	Decrypt Encrypted & Signed File Time
CAST-40	~8 sec	~3 sec	~7 sec	~2 sec	~8 sec	~2 sec
CAST-64	~7 sec	~2 sec	~7 sec	~2 sec	~7 sec	~2 sec
CAST-80	~7 sec	~2 sec	~10 sec	~4 sec	~7 sec	~3 sec
CAST-128	~7 sec	~3 sec	~8 sec	~2 sec	~7 sec	~3 sec
IDEA	~10 sec	~4 sec	~7 sec	~2 sec	~10 sec	~5 sec
DES	~9 sec	~3 sec	~8 sec	~3 sec	~9 sec	~4 sec
3DES	~14 sec	~6 sec	~7 sec	~2 sec	~14 sec	~5 sec
RC2-40	~14 sec	~6 sec	~7 sec	~2 sec	~14 sec	~6 sec
RC2-128	~14 sec	~6 sec	~7 sec	~2 sec	~14 sec	~5 sec

Table 3: Testing a Variety of Encryption Technologies [Henson01]

Our results are generally consistent with Hudson's study. DES performed better than RC4 with a 64 bit key and, 3DES exhibited better performance than RC4 with a 192 bit key.

6.2.3 Security Performance

Table 4 shows the digital signature verification times and maximum throughput for two hash functions and various key sizes. The verification time using SHA-1 is between 35 and 38 percent higher than for MD5 in this study [Manasce03]. Our results are consistent with this study. SHA-1 took more time and had less maximum throughput than MD5. Verification times for both SHA1 and MD5 increased with increasing key size and maximum throughput decreased with increasing key size.

Hash function	Public key size (bits)	Verification time (seconds)	Maximum throughput (requests per second)
MD5	512	0.067	14.9
	768	0.068	14.8
	1024	0.068	14.6
	2048	0.073	13.8
SHA-1	512	0.093	10.8
	768	0.093	10.7
	1024	0.094	10.6
	2048	0.098	10.2

Table 4: Digital Signature Verification Times and Maximum Throughput [Menasce03]

Figure 22 shows maximum throughput of secure socket layer for various combinations of cryptographic algorithms and key lengths. Maximum throughput decreased with increasing key size/security and RC4-MD5 had better throughput than 3DES-SHA1. We also studied SSL performance with a security algorithm using 3DES with SHA1 and RC4 with MD5. RC4 with MD5 had better throughput than 3DES with SHA1. The throughput for both algorithms decreased with an increase in RSA key length.

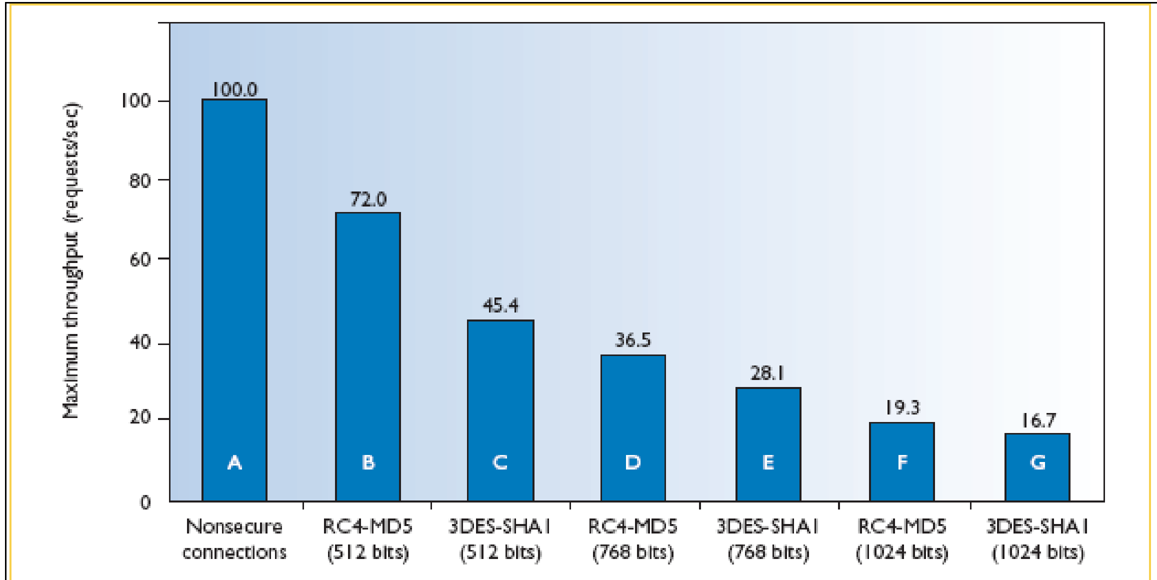

Figure 22: Maximum Throughput of Secure Socket Layer [Menasce03]

6.3 Conclusions

The purpose of this project was to conduct performance analysis of various network security algorithms. It has been shown that the results were generally consistent with other such studies. The results should therefore, be able to be used by application developers when deciding upon the appropriate security algorithm for their applications.

REFERENCES

Print Publications:

[Hook05]
Hook, D., Cryptography with Java, Wiley Publishing, Indianapolis, IN, 2005.

[Tanenbaum03]
Tanenbaum, A., Computer Networks, Prentice Hall, Upper Saddle River, NJ, 2003.

Electronic Sources:

[Henson01]
Henson, T. J., "Testing a Variety of Encryption Technologies," http://www.llnl.gov/tid/lof/documents/pdf/243786.pdf, last revision April 9, 2001, last accessed November 12, 2006.

[Vocal03]
Vocal Technologies, "RC4 Encryption Algorithm," http://www.vocal.com/RC4.pdf, last revision 2003, last accessed November 11, 2006.

[Wei04]
Wei, D., "Crypto++ 5.2.1 Benchmarks," http://www.eskimo.com/~weidai/benchmarks.html, last revision July 23, 2004, last accessed November 12, 2006.

[Menasce03]
Menasce, D., "Security Performance," http://csd.computer.org/dl/mags/ic/2003/03/w3084.htm, Internet Computing (Vol. 7, No. 3) pp. 84-87, May/June, 2003.

[Williams02]
Williams, L., "A Discussion of the Importance of Key Length in Symmetric and Asymmetric Cryptography," http://www.giac.org/certified_professionals/practicals/gsec/0848.php.

[Wikipidea01]
Wikipidea., "Key Size," http://en.wikipedia.org/wiki/Key_size, last revision November 22, 2006.

[Wikipidea02]
Wikipidea., "ElGamal Encryption," http://en.wikipedia.org/wiki/ElGamal_encryption, last revision November 2, 2006.

Appendix A

SECURITY PROTOCOL CODE LISTINGS

```
* * * * * * * * * * * * * * * * * * * * * * *
*                                           *
*  Program name: SecurityServer.java        *
*                                           *
* * * * * * * * * * * * * * * * * * * * * * *

package unf.grad.proj;

import java.io.*;
import java.net.*;

/**
 * @author Praveen Donta
 * File: SecurityServer.java
 * Desc: Security server for accepting client connections and
decrypting encrypted text file.
 *
 */
public class SecurityServer {
  static ServerSocket sr;
  boolean stop = false;

  public static void main(String argv[]) {
      String port;                  // server port
      ServerThread st;
        ServerSocket serverSocket = null;
        boolean listening = true;

        try {

            // Check for input
            if (argv.length >= 1) {
                  port = argv[0];
            } else {
                  System.out.println("Missing server port. Please enter
server port.");
                  port = getString();
            }

            // Start new server socket
            try {
                  serverSocket = new
ServerSocket(Integer.parseInt(port));
            } catch (IOException e) {
                  System.err.println("Could not listen on port:
1234.");
                  System.exit(-1);
            }
```

```
            while (listening)
                  new ServerThread(serverSocket.accept()).start();

            serverSocket.close();

      } catch (IOException e) {
            System.out.println("%s", e.message());
      }
      //final Reader from_server = new
InputStreamReader(s.getInputStream());
      //PrintWriter to_server = new PrintWriter(new
OutputStreamWriter(s.getOutputStream()));
  }

 /**
 * @return string for the input
 * @throws IOException
 */
  public static String getString() throws IOException
  {
       InputStreamReader isr = new InputStreamReader(System.in);
       BufferedReader br = new BufferedReader(isr);
       String s = br.readLine();
       return s; package unf.grad.proj;
```

*** ***
***   ***
*** Program name: ServerThread.java   ***
***   ***
*** ***

```
import java.io.*;
import java.net.*;
import java.util.Calendar;

/**
 * @author Praveen Donta
 * File: SecurityThread.java
 * Desc:
 *
 */
public class ServerThread extends Thread {
  private Socket socket = null;
  private PrintWriter outStream;

  public ServerThread(Socket client) {
        this.socket = client;
  }

  public void run() {

      String inputLine;
      String serverFile = "ServerFile.txt";
      try {

      System.out.println("Client Connected");
```

```
      PrintWriter out = new PrintWriter(socket.getOutputStream(),
true);
      BufferedReader in = new BufferedReader(
                              new InputStreamReader(
                              socket.getInputStream()));
      outStream = new PrintWriter(new FileOutputStream(serverFile));

      out.println("test message");

      while ((inputLine = in.readLine()) != null) {

            if (inputLine.equals("DES"))
                  processDES(in);
            else if (inputLine.equals("3DES"))
                  process3DES(in);
            else if (inputLine.equals("RC4"))
                  processRC4(in);
            else if (inputLine.equals("AES"))
                  processAES(in);
            else if (inputLine.equals("RSA"))
                  processRSA(in);
            else if (inputLine.equals("ELG"))
                  processELG(in);
            else if (inputLine.equals("SHA1"))
                  processSHA1(in);
            else if (inputLine.equals("MD5"))
                  processMD5(in);

            System.out.println("Message from client: " + inputLine);
            if (inputLine == "Bye")
                  break;
            out.println("received");
      }
      } catch (InterruptedIOException e) {}
        catch (IOException e) {}
  }

 /**
 * @return void
 * @throws IOException
 */
  public void processDES(BufferedReader in) {
       String inputLine = null;
       PrintWriter out = null;
       String encryptfile = "DESEncrypt.txt";
       String outfile = "DESDecrypt.txt";
       long sttime, endtime;

       try
       {
            out = new PrintWriter(new FileOutputStream(encryptfile));
       }
       catch (IOException fnf)
       {
            System.err.println("Output file not created
["+encryptfile+"]");
            System.exit(1);
```

```
        }

        try {
             outfile = "DES" + in.readLine();

             while ((inputLine = in.readLine()) != null) {
                   if (inputLine.equals("end"))
                         break;
                   out.write(inputLine);
                   out.write('\n');
             }
        } catch (IOException e) {}

        out.close();

        prDES de = new prDES(encryptfile, outfile, false);

        /*
         * Start and end timer
         */
        sttime = startTimer();
        de.process();
        endtime = endTimer();
        System.out.println("Time took for Decryption of DES " + outfile
+ " : " + (endtime - sttime));
        outStream.println("Time took for Decryption of DES " + outfile +
" : " + (endtime - sttime));
        outStream.flush();
  }

 /**
 * @return void
 * @throws IOException
 */
  public void process3DES(BufferedReader in) {
       String inputLine = null;
       PrintWriter out = null;
       String encryptfile = "3DESEncrypt.txt";
       String outfile = "3DESDecrypt.txt";
       long sttime, endtime;

       try
       {
          out = new PrintWriter(new FileOutputStream(encryptfile));
       }
       catch (IOException fnf)
       {
          System.err.println("Output file not created
["+encryptfile+"]");
          System.exit(1);
       }

       try {
          outfile = "3DES" + in.readLine();
          while ((inputLine = in.readLine()) != null) {
                if (inputLine.equals("end"))
                      break;
```

```
                out.write(inputLine);
                out.write('\n');
            }
        } catch (IOException e) {}

        out.close();

        pr3DES de = new pr3DES(encryptfile, outfile, false);

        /*
         * Start and end timer
         */
        sttime = startTimer();
        de.process();
        endtime = endTimer();
        System.out.println("Time took for Decryption of 3DES " + outfile
+ " : " + (endtime - sttime));
        outStream.println("Time took for Decryption of 3DES " + outfile +
" : " + (endtime - sttime));
        outStream.flush();
    }

   /**
   * @return void
   * @throws IOException
   */
    public void processRC4(BufferedReader in) {
        String inputLine = null;
        String keylen = null;
        PrintWriter out = null;
        String encryptfile = "Rc4Encrypt.txt";
        String outfile = "RC4Decrypt.txt";
        long sttime, endtime;

        try
        {
           out = new PrintWriter(new FileOutputStream(encryptfile));
        }
        catch (IOException fnf)
        {
           System.err.println("Output file not created
["+encryptfile+"]");
           System.exit(1);
        }

        try {
           outfile = in.readLine();
           keylen = in.readLine();
           outfile = "RC4" + keylen + outfile;

           while ((inputLine = in.readLine()) != null) {
                 if (inputLine.equals("end"))
                       break;
                 out.write(inputLine);
                 out.write('\n');
           }
        } catch (IOException e) {}
```

```
      out.close();

      prRC4 rc = new prRC4(encryptfile, outfile, keylen, false);

      /*
       * Start and end timer
       */
      sttime = startTimer();
      rc.process();
      endtime = endTimer();
      System.out.println("Time took for Decryption of RC4 " + outfile +
"of keylen " + keylen + " :" + (endtime - sttime));
    outStream.println("Time took for Decryption of RC4 " + outfile +
"of keylen " + keylen + " :" + (endtime - sttime));
    outStream.flush();
  }


 /**
 * @return void
 * @throws IOException
 */
  public void processAES(BufferedReader in) {
      String inputLine = null;
      String keylen = null;
      PrintWriter out = null;
      String encryptfile = "AESEncrypt.txt";
      String outfile = "AESDecrypt.txt";
      long sttime, endtime;

      try
      {
         out = new PrintWriter(new FileOutputStream(encryptfile));
      }
      catch (IOException fnf)
      {
         System.err.println("Output file not created
["+encryptfile+"]");
         System.exit(1);
      }

      try {
         outfile = in.readLine();
         keylen = in.readLine();
         outfile = "AESOut" + keylen + outfile;

         while ((inputLine = in.readLine()) != null) {
               if (inputLine.equals("end"))
                     break;
               out.write(inputLine);
               out.write('\n');
         }
      } catch (IOException e) {}

      out.close();
```

```
      prAES ae = new prAES(encryptfile, outfile, keylen, false);

      /*
       * Start and end timer
       */
      sttime = startTimer();
      ae.process();
      endtime = endTimer();
      System.out.println("Time took for Decryption of AES " + outfile +
"of keylen " + keylen + " :" + (endtime - sttime));
    outStream.println("Time took for Decryption of AES " + outfile +
"of keylen " + keylen + " :" + (endtime - sttime));
    outStream.flush();
  }

 /**
 * @return void
 * @throws IOException
 */
  public void processRSA(BufferedReader in) {
       String inputLine = null;
       PrintWriter out = null;
       String encryptfile = "RSAEncrypt.txt";
       String outfile = "RSADecrypt.txt";
       long sttime, endtime;

       try
       {
            out = new PrintWriter(new FileOutputStream(encryptfile));
       }
       catch (IOException fnf)
       {
            System.err.println("Output file not created
["+encryptfile+"]");
            System.exit(1);
       }

       try {
            outfile = "RSAOut" + in.readLine();

            readRsaKeyStream(in);

            while ((inputLine = in.readLine()) != null) {
                  if (inputLine.equals("end"))
                        break;
                  out.write(inputLine);
                  out.write('\n');
            }
       } catch (IOException e) {}

       out.close();

       prRSA rsa = new prRSA(encryptfile, outfile, false);

       /*
        * Start and end timer
        */
```

```
        sttime = startTimer();
        rsa.process("");
        endtime = endTimer();
        System.out.println("Time took for Decryption of RSA " + outfile
+ " :" + (endtime - sttime));
      outStream.println("Time took for Decryption of RSA " + outfile + "
:" + (endtime - sttime));
      outStream.flush();
  }

 /**
 * @return void
 * @throws IOException
 */
  public void processELG(BufferedReader in) {
      String inputLine = null;
      PrintWriter out = null;
      String encryptfile = "ELGEncrypt.txt";
      String outfile = "ELGDecrypt.txt";
      long sttime, endtime;

      try
      {
         out = new PrintWriter(new FileOutputStream(encryptfile));
      }
      catch (IOException fnf)
      {
         System.err.println("Output file not created
["+encryptfile+"]");
         System.exit(1);
      }

      try {
         outfile = "ELGOut" + in.readLine();
         readElgKeyStream(in);
         while ((inputLine = in.readLine()) != null) {
               if (inputLine.equals("end"))
                     break;
               out.write(inputLine);
               out.write('\n');
         }
      } catch (IOException e) {}

      out.close();

      prElGamal elg = new prElGamal(encryptfile, outfile, false);

      /*
       * Start and end timer
       */
      sttime = startTimer();
      elg.process("");
      endtime = endTimer();
      System.out.println("Time took for Decryption of AlGamal " +
outfile + " :" + (endtime - sttime));
      outStream.println("Time took for Decryption of AlGamal " +
outfile + " :" + (endtime - sttime));
```

```
        outStream.flush();
    }

   /**
   * @return void
   * @throws IOException
   */
    public void processSHA1(BufferedReader in) {
            String inputLine = null;
            PrintWriter out = null;
            PrintWriter encrypt = null;
            String encryptmd = null;
            String clientmd = null;
            String servmd = null;
            String infile = "SHA1file.txt";
            String encryptfile = "RSAEncrypt.txt";
            String outfile = "RSADecrypt.txt";
            long sttime, endtime;
            byte[] md;

            try
            {
                 out = new PrintWriter(new FileOutputStream(infile));
                 encrypt = new PrintWriter(new
FileOutputStream(encryptfile));
            }
            catch (IOException fnf)
            {
                 System.err.println("Output file not created
["+infile+"]");
                 System.exit(1);
            }

            try {
                 /*
                  * Input file name
                  */
                 outfile = "SHA1Out" + in.readLine();

                 /*
                  * Get Client Message Digest
                  */
                 clientmd = in.readLine();
                 /*
                  * Get RSA Keys
                  */
                 readRsaKeyStream(in);
                 /*
                  * encrypted RSA message digest
                  */
                 encryptmd = in.readLine();
                 encrypt.write(encryptmd);

                 /*
                  * Get Input file
                  */
                 while ((inputLine = in.readLine()) != null) {
```

```
                    if (inputLine.equals("end"))
                          break;
                    out.write(inputLine);
                    out.write('\n');
              }
         } catch (IOException e) {}

         out.close();
         encrypt.close();

         /*
          * Get Message Digest for input file
          */
         prSHA1 sha1 = new prSHA1(infile);

         /*
            * Start and end timer
            */
         sttime = startTimer();
         md = sha1.process();
         endtime = endTimer();
         System.out.println("Time took for SHA1 MessageDigest " +
outfile + " :" + (endtime - sttime));
       outStream.println("Time took for SHA1 MessageDigest " + outfile
+ " :" + (endtime - sttime));
       outStream.flush();
         /*
          * Decrypt RSA message
          */
         prRSA rsa = new prRSA(encryptfile, outfile, false);

         /*
            * Start and end timer
            */
         sttime = startTimer();
         rsa.process("");
         endtime = endTimer();
         System.out.println("Time took for Decryption of RSA " +
outfile + " :" + (endtime - sttime));
         outStream.println("Time took for Decryption of RSA " + outfile
+ " :" + (endtime - sttime));
         outStream.flush();

         /*
          * Compare results
          */
         try {
            BufferedInputStream mdstream =
                  new BufferedInputStream(new
FileInputStream(outfile));
            BufferedReader mdReader = new BufferedReader(new
InputStreamReader(mdstream));
            servmd = mdReader.readLine();
          mdstream.close();
          mdReader.close();
         } catch (IOException e){}
```

```
         if (servmd.equals(new String(md)))
                  System.out.println("File is Authenticated
successfully");
         else
            System.out.println("File Authentication failed");
  }

 /**
 * @return void
 * @throws IOException
 */
  public void processMD5(BufferedReader in) {
         String inputLine = null;
         PrintWriter out = null;
         PrintWriter encrypt = null;
         String encryptmd = null;
         String clientmd = null;
         String servmd = null;
         String infile = "MD5file.txt";
         String encryptfile = "RSAEncrypt.txt";
         String outfile = "RSADecrypt.txt";
         long sttime, endtime;
         byte[] md;

         try
         {
              out = new PrintWriter(new FileOutputStream(infile));
              encrypt = new PrintWriter(new
FileOutputStream(encryptfile));
         }
         catch (IOException fnf)
         {
              System.err.println("Output file not created
["+infile+"]");
              System.exit(1);
         }

         try {
              /*
               * Input file name
               */
              outfile = "MD5Out" + in.readLine();

              /*
               * Get Client Message Digest
               */
              clientmd = in.readLine();
              /*
               * Get RSA Keys
               */
              readRsaKeyStream(in);
              /*
               * encrypted RSA message digest
               */
              encryptmd = in.readLine();
              encrypt.write(encryptmd);
```

```
                /*
                 * Get Input file
                 */
                while ((inputLine = in.readLine()) != null) {
                      if (inputLine.equals("end"))
                            break;
                      out.write(inputLine);
                      out.write('\n');
                }
          } catch (IOException e) {}

          out.close();
          encrypt.close();

          /*
           * Get Message Digest for input file
           */
          prMD5 md5 = new prMD5(infile);

          /*
             * Start and end timer
             */
          sttime = startTimer();
          md = md5.process();
          endtime = endTimer();
          System.out.println("Time took for MD5 Message Digest " +
outfile + " :" + (endtime - sttime));
        outStream.println("Time took for MD5 Message Digest " + outfile
+ " :" + (endtime - sttime));
        outStream.flush();

          /*
           * Decrypt RSA message
           */
          prRSA rsa = new prRSA(encryptfile, outfile, false);

          /*
             * Start and end timer
             */
          sttime = startTimer();
          rsa.process("");
          endtime = endTimer();
          System.out.println("Time took for Decryption of RSA " +
outfile + " :" + (endtime - sttime));
          outStream.println("Time took for Decryption of RSA " + outfile
+ " :" + (endtime - sttime));
          outStream.flush();

          /*
           * Compare results
           */
          try {
             BufferedInputStream mdstream =
                   new BufferedInputStream(new
FileInputStream(outfile));
             BufferedReader mdReader = new BufferedReader(new
InputStreamReader(mdstream));
```

```
            servmd = mdReader.readLine();
          mdstream.close();
          mdReader.close();
         } catch (IOException e){}

         if (servmd.equals(new String(md)))
                  System.out.println("File is Authenticated
successfully");
         else
            System.out.println("File Authentication failed");
  }

 /**
 * @return void
 * @throws IOException
 */
  private static void readRsaKeyStream(BufferedReader in)
      {
        String bv;
        String keyfile = "ServRSAKey.dat";
        PrintWriter out;
        int i;
        try {
              out = new PrintWriter(new FileOutputStream(keyfile));

              // write variables mod, pubExp, privExp, p, q, dp, dq,
qinv
              for (i=0; i < 8;i++)
                  if ((bv = in.readLine()) != null)
                     out.println(bv);

              out.close();
        } catch (IOException createKey)
        {
              System.err.println("Could not decryption create key file
"+ "["+keyfile+"]");
              System.exit(1);
        }
      }

 /**
 * @return void
 * @throws IOException
 */
      private static void readElgKeyStream(BufferedReader in)
      {
        String bv;
        String keyfile = "ServElGamalKey.dat";
        PrintWriter out;
        int i;

        try {
              out = new PrintWriter(new FileOutputStream(keyfile));

              // write variables p, g, x
              for (i=0; i < 3;i++)
                    if ((bv = in.readLine()) != null)
```

```
                            out.println(bv);

                out.close();

            } catch (IOException createKey)
            {
                System.err.println("Could not decryption create key file
"+
"["+keyfile+"]");
                System.exit(1);
            }
        }

    /**
    * @return long
    * @throws IOException
    */
     public static long startTimer()
     {
         Calendar rightNow;
         String outTxt;
         String addr;
         FileWriter out;
         long sttime;
         int hour, min, sec, milli;

         //    Get Starting Time
         rightNow = Calendar.getInstance();
         sttime = System.nanoTime();
         hour = rightNow.get(Calendar.HOUR);
         min = rightNow.get(Calendar.MINUTE);
         sec = rightNow.get(Calendar.SECOND);
         milli = rightNow.get(Calendar.MILLISECOND);

       // Display Results
         System.out.println("Starting Time for server decryption "
                      + Integer.toString(hour) + ":"
                      + Integer.toString(min) + ":"
                      + Integer.toString(sec) + ":"
                      + Integer.toString(milli) + "\n");

         return sttime;
     }

    /**
    * @return long
    * @throws IOException
    */
     public static long endTimer()
     {
         Calendar rightNow;
         String outTxt;
         String addr;
         FileWriter out;
         long endtime;
         int hour, min, sec, milli;
```

```
      //    Get Ending Time
      rightNow = Calendar.getInstance();
      endtime = System.nanoTime();
      hour = rightNow.get(Calendar.HOUR);
      min = rightNow.get(Calendar.MINUTE);
      sec = rightNow.get(Calendar.SECOND);
      milli = rightNow.get(Calendar.MILLISECOND);

      // Display Results
      System.out.println("Ending Time for server decryption "
                  + Integer.toString(hour) + ":"
                  + Integer.toString(min) + ":"
                  + Integer.toString(sec) + ":"
                  + Integer.toString(milli) + "\n");

      return endtime;
  }

 /**
 * @return string for the input
 * @throws IOException
 */
  public static String getString() throws IOException
  {
       InputStreamReader isr = new InputStreamReader(System.in);
       BufferedReader br = new BufferedReader(isr);
       String s = br.readLine();
       return s;
  }

 /**
 * @return long representation for the input
 * @throws IOException
 */
  public static long getLong() throws IOException
  {
       String s = getString();
       return Long.parseLong(s);
  }
}  }

 /**
 * @return long representation for the input
 * @throws IOException
 */
  public static long getLong() throws IOException
  {
       String s = getString();
       return Long.parseLong(s);
  }
}

* * * * * * * * * * * * * * * * * * * * * * *
*                                             *
*  Program name: SecurityClient.java          *
*                                             *
```

* *

```
package unf.grad.proj;

import java.io.*;
import java.net.*;
import java.util.Calendar;

/**
 * @author Praveen Donta
 * File: SecurityClient.java
 * Desc: Security Client connects to server and encrypts a text file
and send it to the server.
 *
 */
public class SecurityClient {
  public static void main(String argv[]) {
    try {
      String server;          // server name
      String port;                  // server port
      String fromServer;
      String infile = "infile1.txt";
      String outfile = "outfile1.txt";
      String mdfile = "mdfile.txt";
      String clientfile = "clientout.txt";
      String keylen = null;
      String rv;
      String outString;
      long ll_in = 0;
      long sttime, endtime;
      byte[] md;
      Socket socket;
        PrintWriter out = null;
        BufferedReader in = null;
        BufferedInputStream inStream;
        PrintWriter outStream = null;

      if (argv.length == 1) {
             server = argv[0];
             System.out.println("Missing server port. Please enter
server port.");
             port = getString();
      } else if (argv.length == 2){
             server = argv[0];
             port = argv[1];
      } else {
            System.out.println("Missing server name. Please enter
server name.");
            server = getString();
            System.out.println("Missing server port. Please enter
server port.");
            port = getString();
      }

      // Create Socket
      socket = new Socket(server, Integer.parseInt(port));
```

```
        out = new PrintWriter(socket.getOutputStream(), true);
        in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

        try
        {
                  outStream = new PrintWriter(new
FileOutputStream(clientfile));
        }
        catch (IOException fnf)
        {
                  System.err.println("Output file not created
["+clientfile+"]");
            System.exit(1);
      }

        while ((fromServer = in.readLine()) != null) {
            System.out.println("Server: " + fromServer);
            System.out.println("1. DES Test - Multiple file sizes with
64 bit key ");
            System.out.println("2. 3DES Test - Multiple file sizes with
192 bit key ");
            System.out.println("3. RC4 Test - Multiple file sizes with
192 bit key");
            System.out.println("4. AES Test - Multiple file sizes with
192 bit key");
            System.out.println("5. RC4 Test - Multiple key sizes with
5K or 10K file");
            System.out.println("6. AES Test - Multiple key sizes with
5K or 10K file");
            System.out.println("7. RSA Test - Multiple key sizes (1024,
2048, 3072, 4096, 5120)");
            System.out.println("8. ElGamal Test - Multiple key sizes
(100, 800, 1000, 1200) ");
            System.out.println("9. RSA with SHA-1 - multiple keys (512,
768, 1024, 2048) bits ");
            System.out.println("10. RSA with MD-5 - multiple keys with
(512, 768, 1024, 2048) bits");
            System.out.println("0. Quit");
            System.out.print("Please choose one of the option: ");

            try {
                  ll_in = getLong();                  // get input from
user
            } catch(NumberFormatException e) {  // catch invalid
integer errors
            System.out.println("\nPlease enter selection from 1 to 10
only, please try again.");
            }
            /************************************/
            /* Option 1 */
            /***********************************/
            if (ll_in == 1) {
                  System.out.println("Enter testfile from 1K to 10K");
                  infile = getString();
                  prDES de = new prDES(infile, outfile, true);
                  /*
```

```
                 * Start and end timer
                 */
                sttime = startTimer();
                de.process();
                endtime = endTimer();
                System.out.println("Time took for encryption of DES
for input file " + infile + " : " + (endtime - sttime));
                outString = "Time took for encryption of DES for
input file " + infile + " : " + (endtime - sttime);
                outStream.println(outString);
                outStream.flush();

                inStream = new BufferedInputStream(new
FileInputStream(outfile));
                BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                out.println("DES");
                out.println(infile);
                while ((rv = br.readLine()) != null)
                      out.println(rv);
                out.println("end");
                br.close();
                inStream.close();
            } else if (ll_in == 2) {
                System.out.println("Enter testfile from 1K to 10K");
                infile = getString();
                pr3DES de = new pr3DES(infile, outfile, true);

                /*
                 * Start and end timer
                 */
                sttime = startTimer();
                de.process();
                endtime = endTimer();
                System.out.println("Time took for encryption of 3DES
for input file " + infile + " : " + (endtime - sttime));
                outString = "Time took for encryption of 3DES for
input file " + infile + " : " + (endtime - sttime);
                outStream.println(outString);
                outStream.flush();

                inStream = new BufferedInputStream(new
FileInputStream(outfile));
                BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                out.println("3DES");
                out.println(infile);
                while ((rv = br.readLine()) != null)
                      out.println(rv);
                out.println("end");
                br.close();
                inStream.close();
            } else if (ll_in == 3) {
                System.out.println("Enter testfile from 1K to 10K");
                infile = getString();
                prRC4 rc = new prRC4(infile, outfile, "3", true);
```

```
                    /*
                     * Start and end timer
                     */
                    sttime = startTimer();
                    rc.process();
                    endtime = endTimer();
                    System.out.println("Time took for encryption of RC4
for input file " + infile + " : " + (endtime - sttime));
                    outString = "Time took for encryption of RC4 for
input file " + infile + " : " + (endtime - sttime);
                    outStream.println(outString);
                    outStream.flush();

                    inStream = new BufferedInputStream(new
FileInputStream(outfile));
                    BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                    out.println("RC4");
                    out.println(infile);
                    out.println("3");
                    while ((rv = br.readLine()) != null)
                          out.println(rv);
                    out.println("end");
                    br.close();
                    inStream.close();
              } else if (ll_in == 4) {
                    System.out.println("Enter testfile from 1K to 10K");
                    infile = getString();
                    prAES ae = new prAES(infile, outfile, "3", true);

                    /*
                     * Start and end timer
                     */
                    sttime = startTimer();
                    ae.process();
                    endtime = endTimer();
                    System.out.println("Time took for encryption of AES
for input file " + infile + " : " + (endtime - sttime));
                    outString = "Time took for encryption of AES for
input file " + infile + " : " + (endtime - sttime);
                    outStream.println(outString);
                    outStream.flush();

                    inStream = new BufferedInputStream(new
FileInputStream(outfile));
                    BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                    out.println("AES");
                    out.println(infile);
                    out.println("3");
                    while ((rv = br.readLine()) != null)
                          out.println(rv);
                    out.println("end");
                    br.close();
                    inStream.close();
              } else if (ll_in == 5) {
                    System.out.println("Enter testfile 5K or 10K");
```

```
                  infile = getString();
                  System.out.println("Enter Key length (2 for 64 bits,
3 for 128 bits, 4 for 256 bits)");
                  keylen = getString();
                  prRC4 rc = new prRC4(infile, outfile, keylen, true);

                  /*
                   * Start and end timer
                   */
                  sttime = startTimer();
                  rc.process();
                  endtime = endTimer();
                  System.out.println("Time took for encryption of RC4
for input file " + infile + " with key len "  + keylen + " : " +
(endtime - sttime));
                  outString = "Time took for encryption of RC4 for
input file " + infile + " with key len "  + keylen + " : " + (endtime -
sttime);
                  outStream.println(outString);
                  outStream.flush();

                  inStream = new BufferedInputStream(new
FileInputStream(outfile));
                  BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                  out.println("RC4");
                  out.println(infile);
                  out.println(keylen);
                  while ((rv = br.readLine()) != null)
                        out.println(rv);
                  out.println("end");
                  br.close();
                  inStream.close();
            } else if (ll_in == 6) {
                  System.out.println("Enter testfile 5K or 10K");
                  infile = getString();
                  System.out.println("Enter Key length (2 for 64 bits,
3 for 128 bits, 4 for 256 bits)");
                  keylen = getString();
                  prAES ae = new prAES(infile, outfile, keylen, true);

                  /*
                   * Start and end timer
                   */
                  sttime = startTimer();
                  ae.process();
                  endtime = endTimer();
                  System.out.println("Time took for encryption of AES
for input file " + infile + " with keylen " + keylen +" : " + (endtime
- sttime));
                  outString = "Time took for encryption of AES for
input file " + infile + " with keylen " + keylen +" : " + (endtime -
sttime);
                  outStream.println(outString);
                  outStream.flush();
```

```
                  inStream = new BufferedInputStream(new
FileInputStream(outfile));
                  BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                  out.println("AES");
                  out.println(infile);
                  out.println(keylen);
                  while ((rv = br.readLine()) != null)
                        out.println(rv);
                  out.println("end");
                  br.close();
                  inStream.close();
            } else if (ll_in == 7) {
                  System.out.println("Enter testfile of 100bytes or 1K
or 5K or 10K");
                  infile = getString();
                  System.out.println("Enter Key length (1024, 2048,
3072, 4096, 5120 bits)");
                  keylen = getString();
                  prRSA rsa = new prRSA(infile, outfile, true);

                  /*
                   * Start and end timer
                   */
                  sttime = startTimer();
                  rsa.process(keylen);
                  endtime = endTimer();
                  System.out.println("Time took for encryption of RSA
for input file " +  infile + " with key len " + keylen + " : " +
(endtime - sttime));
                  outString = "Time took for encryption of RSA for
input file " +  infile + " with key len " + keylen + " : " + (endtime -
sttime);
                  outStream.println(outString);
                  outStream.flush();

                  out.println("RSA");
                  out.println(infile);
                  writeRsaKeyStream(out);
                  inStream = new BufferedInputStream(new
FileInputStream(outfile));
                  BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                  while ((rv = br.readLine()) != null)
                        out.println(rv);
                  out.println("end");
                  br.close();
                  inStream.close();
            } else if (ll_in == 8) {
                  System.out.println("Enter testfile of 100bytes or 1K
or 5K or 10K");
                  infile = getString();
                  System.out.println("Enter Key length (100, 800, 1000,
1200 bits)");
                  keylen = getString();
                  prElGamal el = new prElGamal(infile, outfile, true);
```

```
                  /*
                   * Start and end timer
                   */
                  sttime = startTimer();
                  el.process(keylen);
                  endtime = endTimer();
                  System.out.println("Time took for encryption of
AlGamal for input file " + infile + " with key len  " + keylen + " :" +
(endtime - sttime));
                  outString = "Time took for encryption of AlGamal for
input file " + infile + " with key len  " + keylen + " :" + (endtime -
sttime);
                  outStream.println(outString);
                  outStream.flush();

                  out.println("ELG");
                  out.println(infile);
                  writeElgKeyStream(out);
                  inStream = new BufferedInputStream(new
FileInputStream(outfile));
                  BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                  while ((rv = br.readLine()) != null)
                        out.println(rv);
                  out.println("end");
                  br.close();
                  inStream.close();
            } else if (ll_in == 9) {
                  /*
                   * Get required Files
                   */
                  System.out.println("Enter testfile of 1 MB");
                  infile = getString();
                  /*
                   * Get Key Strength
                   */
                  System.out.println("Enter Key length (512, 768, 1024,
2048 bits)");
                  keylen = getString();

                  /*
                   * Get Message Digest for given input file and write
to a file for RSA encryption
                   */
                  prSHA1 sha = new prSHA1(infile);
                  /*
                   * Start and end timer
                   */
                  sttime = startTimer();
                  md = sha.process();
                  endtime = endTimer();
                  System.out.println("Time took for SHA1 message digest
for input file " + infile + " : " + (endtime - sttime));
                  outString = "Time took for SHA1 message digest for
input file " + infile + " : " + (endtime - sttime);
                  outStream.println(outString);
                  outStream.flush();
```

```
                    writeToFile(mdfile, new String(md));

                    /*
                     * Encrypt Message digest with RSA
                     */
                    prRSA rsa = new prRSA(mdfile, outfile, true);
                    /*
                     * Start and end timer
                     */
                    sttime = startTimer();
                    rsa.process(keylen);
                    endtime = endTimer();
                    System.out.println("Time took for encryption of RSA
with keylen " + keylen + " : " + (endtime - sttime));
                    outString = "Time took for encryption of RSA with
keylen " + keylen + " : " + (endtime - sttime);
                    outStream.println(outString);
                    outStream.flush();

                    /*
                     * Write stream to Server socket
                     */
                    out.println("SHA1");    /* Algorithm */
                    out.println(infile);    /* input file */
                    out.println(new String(md));        /* message digest
*/
                    writeRsaKeyStream(out);  /* write RSA keys */

                    /*
                     * write RSA encrytion
                     */
                    inStream = new BufferedInputStream(new
FileInputStream(outfile));
                    BufferedReader rs = new BufferedReader(new
InputStreamReader(inStream));
                    out.println(rs.readLine());
                    rs.close();
                    inStream.close();

                    /*
                     * Write input file
                     */
                    inStream = new BufferedInputStream(new
FileInputStream(infile));
                    BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                    while ((rv = br.readLine()) != null)
                          out.println(rv);

                    /*
                     * End of communication
                     */
                    out.println("end");
                    br.close();
                    inStream.close();
          }  else if (ll_in == 10) {
```

```
                /*
                 * Get Required file
                 */
                System.out.println("Enter testfile of 1 MB");
                infile = getString();
                /*
                 * Get Key length
                 */
                System.out.println("Enter Key length (512, 768, 1024, 
2048 bits)");
                keylen = getString();

                /*
                 * process message digest and write to a file
                 */
                prMD5 md5 = new prMD5(infile);

                /*
                   * Start and end timer
                   */
                sttime = startTimer();
                md = md5.process();
                endtime = endTimer();
                System.out.println("Time took for MD5 Message Digest for 
input file " + infile + " : " + (endtime - sttime));
                outString = "Time took for MD5 Message Digest for input 
file " + infile + " : " + (endtime - sttime);
                outStream.println(outString);
                outStream.flush();

                writeToFile("", new String(md));
                prRSA rsa = new prRSA(mdfile, outfile, true);

                /*
                 * Start and end timer
                 */
                sttime = startTimer();
                rsa.process(keylen);
                endtime = endTimer();
                System.out.println("Time took for encryption of RSA with 
keylen " + keylen + " : " + (endtime - sttime));
                outString = "Time took for encryption of RSA with keylen 
" + keylen + " : " + (endtime - sttime);
                outStream.println(outString);
                outStream.flush();

                /*
                 * Write stream to a socket server
                 */
                out.println("MD5");
                out.println(infile);
                out.println(md);
                writeRsaKeyStream(out);

                /*
                   * write RSA encrytion
                   */
```

```
                inStream = new BufferedInputStream(new
FileInputStream(outfile));
                BufferedReader rs = new BufferedReader(new
InputStreamReader(inStream));
                out.println(rs.readLine());
                rs.close();
                inStream.close();

                /*
                   * Write input file
                   */
                inStream = new BufferedInputStream(new
FileInputStream(infile));
                BufferedReader br = new BufferedReader(new
InputStreamReader(inStream));
                while ((rv = br.readLine()) != null)
                      out.println(rv);

                /*
                   * End of communication
                   */
                out.println("end");
                br.close();
                inStream.close();
          } else if (ll_in == 0) {
                 outStream.close();
                   return;
             /****************************/
             /* Invalid option **********/
             /***************************/
             } else {
                   System.out.println("Invalid entry. Please select a
selection from 1 to 10");
                   System.out.println("\n");
             }
         }
     }
     catch (Exception e) {
       System.out.println("Error while looking up account:");
       e.printStackTrace();
     }
   }

  /**
  * @return void
  * @throws IOException
  */
   private static void writeRsaKeyStream(PrintWriter out)
   {
       String bv;
       String keyfile = "RSAKey.dat";
       int i;
       try {
             BufferedInputStream keystream =
                   new BufferedInputStream(new
FileInputStream(keyfile));
```

```
            BufferedReader keyReader = new BufferedReader(new
InputStreamReader(keystream));

            // write variables mod, pubExp, privExp, p, q, dp, dq, qinv
            for (i=0; i < 8;i++)
              if ((bv = keyReader.readLine()) != null)
                   out.println(bv);

            keystream.close();
            keyReader.close();

      } catch (IOException createKey)     {
            System.err.println("Could not decryption create key file "+
"["+keyfile+"]");
            System.exit(1);
      }
  }

 /**
 * @return void
 * @throws IOException
 */
  private static void writeElgKeyStream(PrintWriter out)
  {
      String bv;
      String keyfile = "ElGamalKey.dat";
      int i;

      try {
            BufferedInputStream keystream =
                  new BufferedInputStream(new
FileInputStream(keyfile));
            BufferedReader keyReader = new BufferedReader(new
InputStreamReader(keystream));

            // write variables p, g, x
            for (i=0; i < 3;i++)
                  if ((bv = keyReader.readLine()) != null)
                        out.println(bv);

            keystream.close();
            keyReader.close();

      } catch (IOException createKey)
      {
            System.err.println("Could not decryption create key file "+
"["+keyfile+"]");
            System.exit(1);
      }
  }

 /**
 * @return void
 * @throws IOException
 */
  public static void writeToFile(String infile, String md)
  {
```

```
      try {
            PrintWriter pr = new PrintWriter(new
FileOutputStream(infile));
            pr.println(md);
            pr.close();
      } catch (IOException createKey) {
      }

  }

  /**
  * @return long representation for the input
  * @throws IOException
  */
   public static long startTimer()
   {
       Calendar rightNow;
       String outTxt;
       String addr;
       FileWriter out;
       long sttime;
       int hour, min, sec, milli;

       //    Get Starting Time
       rightNow = Calendar.getInstance();
       sttime = System.nanoTime();
       hour = rightNow.get(Calendar.HOUR);
       min = rightNow.get(Calendar.MINUTE);
       sec = rightNow.get(Calendar.SECOND);
       milli = rightNow.get(Calendar.MILLISECOND);

         // Display Results
       System.out.println("Starting Time for client encrytion"
                    + Integer.toString(hour) + ":"
                    + Integer.toString(min) + ":"
                    + Integer.toString(sec) + ":"
                    + Integer.toString(milli) + "\n");

       return sttime;
   }

  /**
  * @return long representation for the input
  * @throws IOException
  */
   public static long endTimer()
   {
       Calendar rightNow;
       String outTxt;
       String addr;
       FileWriter out;
       long endtime;
       int hour, min, sec, milli;

       //    Get Ending Time
       rightNow = Calendar.getInstance();
       endtime = System.nanoTime();
```

```
      hour = rightNow.get(Calendar.HOUR);
      min = rightNow.get(Calendar.MINUTE);
      sec = rightNow.get(Calendar.SECOND);
      milli = rightNow.get(Calendar.MILLISECOND);

      // Display Results
      System.out.println("Ending Time for client decryption"
                  + Integer.toString(hour) + ":"
                  + Integer.toString(min) + ":"
                  + Integer.toString(sec) + ":"
                  + Integer.toString(milli) + "\n");

      return endtime;
  }

 /**
 * @return string for the input
 * @throws IOException
 */
  public static String getString() throws IOException
  {
       InputStreamReader isr = new InputStreamReader(System.in);
       BufferedReader br = new BufferedReader(isr);
       String s = br.readLine();
       return s;
  }

 /**
 * @return long representation for the input
 * @throws IOException
 */
  public static long getLong() throws IOException
  {
       String s = getString();
       return Long.parseLong(s);
  }
}
```

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                 *
*  Program name: prDES.java                                       *
*                                                                 *
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
```

```
/**
 * @author Praveen Donta
 * File: prDES.java
 * Desc: encrypts and decrypts text file using  DES algorithm.
 *
 */

package unf.grad.proj;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.FileInputStream;
```

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.crypto.engines.DESEngine;
import org.bouncycastle.crypto.modes.CBCBlockCipher;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.util.encoders.Hex;

public class prDES extends Object
{
    // Encrypting or decrypting ?
    private boolean encrypt = true;

    // To hold the initialised DESede cipher
    private PaddedBufferedBlockCipher cipher = null;

    // The input stream of bytes to be processed for encryption
    private BufferedInputStream in = null;

    // The output stream of bytes to be procssed
    private BufferedOutputStream out = null;

    // The key
    private byte[] key = null;

    /*
     * start the application
     */
    public static void main(String[] args)
    {
        boolean encrypt = true;
        String infile = null;
        String outfile = null;

        if (args.length < 2)
        {
            prDES de = new prDES();
            System.err.println("Usage: java "+de.getClass().getName()+
                                    " infile outfile");
            System.exit(1);
        }
            infile = args[0];
            outfile = args[1];

            if (args.length > 2)
            {
                encrypt = false;
        }

        prDES de = new prDES(infile, outfile, encrypt);
        de.process();
    }
```

```
    // Default constructor, used for the usage message
    public prDES()
    {
    }

    /*
     * Constructor, that takes the arguments appropriate for
     * processing the command line directives.
     */
    public prDES(
                String infile,
                String outfile,
                boolean encrypt)
    {
        /*
         * First, determine that infile & keyfile exist as appropriate.
         *
         * This will also create the BufferedInputStream as required
         * for reading the input file. All input files are treated
         * as if they are binary, even if they contain text, it's the
         * bytes that are encrypted.
         */
        this.encrypt = encrypt;
        try
        {
            in = new BufferedInputStream(new FileInputStream(infile));
        }
        catch (FileNotFoundException fnf)
        {
            System.err.println("Input file not found ["+infile+"]");
            System.exit(1);
        }

        try
        {
            out = new BufferedOutputStream(new
FileOutputStream(outfile));
        }
        catch (IOException fnf)
        {
            System.err.println("Output file not created
["+outfile+"]");
            System.exit(1);
        }

            key = Hex.decode("0123456789abcdef");

    }

    public final void process()
    {
        /*
         * Setup the DESede cipher engine, create a
PaddedBufferedBlockCipher
         * in CBC mode.
         */
        cipher = new PaddedBufferedBlockCipher(
```

```
                                    new CBCBlockCipher(new
DESEngine()));

        /*
         * The input and output streams are currently set up
         * appropriately, and the key bytes are ready to be
         * used.
         *
         */

        if (encrypt)
        {
            performEncrypt(key);
        }
        else
        {
            performDecrypt(key);
        }

        // after processing clean up the files
        try
        {
            in.close();
            out.flush();
            out.close();
        }
        catch (IOException closing)
        {

        }
    }

    /*
     * This method performs all the encryption and writes
     * the cipher text to the buffered output stream created
     * previously.
     */
    private final void performEncrypt(byte[] key)
    {
        // initialise the cipher with the key bytes, for encryption
        cipher.init(true, new KeyParameter(key));

        /*
         * Create some temporary byte arrays for use in
         * encryption, make them a reasonable size so that
         * we don't spend forever reading small chunks from
         * a file.
         *
         */
        // int inBlockSize = cipher.getBlockSize() * 5;
        int inBlockSize = 47;
        int outBlockSize = cipher.getOutputSize(inBlockSize);

        byte[] inblock = new byte[inBlockSize];
        byte[] outblock = new byte[outBlockSize];

        /*
```

```
         * now, read the file, and output the chunks
         */
        try
        {
            int inL;
            int outL;
            byte[] rv = null;
            while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
            {
                outL = cipher.processBytes(inblock, 0, inL, outblock,
0);
                /*
                 * Before we write anything out, we need to make sure
                 * that we've got something to write out.
                 */
                if (outL > 0)
                {
                    rv = Hex.encode(outblock, 0, outL);
                    out.write(rv, 0, rv.length);
                    out.write('\n');
                }
            }

            try
            {
                /*
                 * Now, process the bytes that are still buffered
                 * within the cipher.
                 */
                outL = cipher.doFinal(outblock, 0);
                if (outL > 0)
                {
                    rv = Hex.encode(outblock, 0, outL);
                    out.write(rv, 0, rv.length);
                    out.write('\n');
                }
            }
            catch (CryptoException ce)
            {

            }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

    /*
     * This method performs all the decryption and writes
     * the plain text to the buffered output stream created
     * previously.
     */
    private final void performDecrypt(byte[] key)
    {
        // initialise the cipher for decryption
        cipher.init(false, new KeyParameter(key));
```

```
        /*
         * As the decryption is from our preformatted file,
         * and we know that it's a hex encoded format, then
         * we wrap the InputStream with a BufferedReader
         * so that we can read it easily.
         */
        BufferedReader br = new BufferedReader(new
InputStreamReader(in));

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int outL;
            byte[] inblock = null;
            byte[] outblock = null;
            String rv = null;
            while ((rv = br.readLine()) != null)
            {
                inblock = Hex.decode(rv);
                outblock = new
byte[cipher.getOutputSize(inblock.length)];

                outL = cipher.processBytes(inblock, 0, inblock.length,
                                            outblock, 0);
                /*
                 * Before we write anything out, we need to make sure
                 * that we've got something to write out.
                 */
                if (outL > 0)
                {
                    out.write(outblock, 0, outL);
                }
            }

            try
            {
                /*
                 * Now, process the bytes that are still buffered
                 * within the cipher.
                 */
                outL = cipher.doFinal(outblock, 0);
                if (outL > 0)
                {
                    out.write(outblock, 0, outL);
                }
            }
            catch (CryptoException ce)
            {

            }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
```

```
        }
    }

}
```

```
* * * * * * * * * * * * * * * * * * * * * * *
*                                           *
*  Program name: pr3DES.java                *
*                                           *
* * * * * * * * * * * * * * * * * * * * * * *
```

```
/**
 * @author Praveen Donta
 * File: pr3DES.java
 * Desc: encrypts and decrypts text file using 3DES algorithm.
 *
 */

package unf.grad.proj;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.crypto.engines.DESedeEngine;
import org.bouncycastle.crypto.modes.CBCBlockCipher;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.util.encoders.Hex;

/**
 *
 */
public class pr3DES extends Object
{
    // Encrypting or decrypting ?
    private boolean encrypt = true;

    // To hold the initialised DESede cipher
    private PaddedBufferedBlockCipher cipher = null;

    // The input stream of bytes to be processed for encryption
    private BufferedInputStream in = null;

    // The output stream of bytes to be procssed
    private BufferedOutputStream out = null;

    // The key
    private byte[] key = null;

    /*
```

```
     * start the application
     */
    public static void main(String[] args)
    {
        boolean encrypt = true;
        String infile = null;
        String outfile = null;

        if (args.length < 2)
        {
            pr3DES de = new pr3DES();
            System.err.println("Usage: java "+de.getClass().getName()+
                                    " infile outfile");
            System.exit(1);
        }
            infile = args[0];
            outfile = args[1];

            if (args.length > 2)
            {
                encrypt = false;
        }

        pr3DES de = new pr3DES(infile, outfile, encrypt);
        de.process();
    }

    // Default constructor, used for the usage message
    public pr3DES()
    {
    }

    /*
     * Constructor, that takes the arguments appropriate for
     * processing the command line directives.
     */
    public pr3DES(
                String infile,
                String outfile,
                boolean encrypt)
    {
        /*
         * First, determine that infile & keyfile exist as appropriate.
         *
         * This will also create the BufferedInputStream as required
         * for reading the input file. All input files are treated
         * as if they are binary, even if they contain text, it's the
         * bytes that are encrypted.
         */
        this.encrypt = encrypt;
        try
        {
            in = new BufferedInputStream(new FileInputStream(infile));
        }
        catch (FileNotFoundException fnf)
        {
            System.err.println("Input file not found ["+infile+"]");
```

```
            System.exit(1);
        }

        try
        {
            out = new BufferedOutputStream(new
FileOutputStream(outfile));
        }
        catch (IOException fnf)
        {
            System.err.println("Output file not created
["+outfile+"]");
            System.exit(1);
        }

            key =
Hex.decode("0123456789abcdef0123456789abcdef0123456789abcdef");

    }

    public final void process()
    {
        /*
         * Setup the DESede cipher engine, create a
PaddedBufferedBlockCipher
         * in CBC mode.
         */
        cipher = new PaddedBufferedBlockCipher(
                                    new CBCBlockCipher(new
DESedeEngine()));

        /*
         * The input and output streams are currently set up
         * appropriately, and the key bytes are ready to be
         * used.
         *
         */

        if (encrypt)
        {
            performEncrypt(key);
        }
        else
        {
            performDecrypt(key);
        }

        // after processing clean up the files
        try
        {
            in.close();
            out.flush();
            out.close();
        }
        catch (IOException closing)
        {
```

```
        }
    }

    /*
     * This method performs all the encryption and writes
     * the cipher text to the buffered output stream created
     * previously.
     */
    private final void performEncrypt(byte[] key)
    {
        // initialise the cipher with the key bytes, for encryption
        cipher.init(true, new KeyParameter(key));

        /*
         * Create some temporary byte arrays for use in
         * encryption, make them a reasonable size so that
         * we don't spend forever reading small chunks from
         * a file.
         *
         */
        // int inBlockSize = cipher.getBlockSize() * 5;
        int inBlockSize = 47;
        int outBlockSize = cipher.getOutputSize(inBlockSize);

        byte[] inblock = new byte[inBlockSize];
        byte[] outblock = new byte[outBlockSize];

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int inL;
            int outL;
            byte[] rv = null;
            while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
            {
                outL = cipher.processBytes(inblock, 0, inL, outblock,
0);
                /*
                 * Before we write anything out, we need to make sure
                 * that we've got something to write out.
                 */
                if (outL > 0)
                {
                    rv = Hex.encode(outblock, 0, outL);
                    out.write(rv, 0, rv.length);
                    out.write('\n');
                }
            }

            try
            {
                /*
                 * Now, process the bytes that are still buffered
                 * within the cipher.
                 */
```

```
                outL = cipher.doFinal(outblock, 0);
                if (outL > 0)
                {
                    rv = Hex.encode(outblock, 0, outL);
                    out.write(rv, 0, rv.length);
                    out.write('\n');
                }
            }
            catch (CryptoException ce)
            {

            }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

    /*
     * This method performs all the decryption and writes
     * the plain text to the buffered output stream created
     * previously.
     */
    private final void performDecrypt(byte[] key)
    {
        // initialise the cipher for decryption
        cipher.init(false, new KeyParameter(key));

        /*
         * As the decryption is from our preformatted file,
         * and we know that it's a hex encoded format, then
         * we wrap the InputStream with a BufferedReader
         * so that we can read it easily.
         */
        BufferedReader br = new BufferedReader(new
InputStreamReader(in));

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int outL;
            byte[] inblock = null;
            byte[] outblock = null;
            String rv = null;
            while ((rv = br.readLine()) != null)
            {
                inblock = Hex.decode(rv);
                outblock = new
byte[cipher.getOutputSize(inblock.length)];

                outL = cipher.processBytes(inblock, 0, inblock.length,
                                            outblock, 0);
                /*
                 * Before we write anything out, we need to make sure
```

```
                 * that we've got something to write out.
                 */
                if (outL > 0)
                {
                    out.write(outblock, 0, outL);
                }
            }

            try
            {
                /*
                 * Now, process the bytes that are still buffered
                 * within the cipher.
                 */
                outL = cipher.doFinal(outblock, 0);
                if (outL > 0)
                {
                    out.write(outblock, 0, outL);
                }
            }
            catch (CryptoException ce)
            {

            }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

}
```

*** ***
***   ***
*** Program name: prRC4.java   ***
***   ***
*** ***

```
/**
 * @author Praveen Donta
 * File: prRC4.java
 * Desc: encrypts and decrypts text file using RC4 algorithm.
 *
 */

package unf.grad.proj;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import org.bouncycastle.crypto.engines.RC4Engine;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.util.encoders.Hex;
import org.bouncycastle.crypto.StreamCipher;

public class prRC4 extends Object
{
    // Encrypting or decrypting ?
    private boolean encrypt = true;

    // To hold the initialised DESede cipher
    private StreamCipher cipher = null;

    // The input stream of bytes to be processed for encryption
    private BufferedInputStream in = null;

    // The output stream of bytes to be procssed
    private BufferedOutputStream out = null;

    // The key
    private byte[] key = null;

    /*
     * start the application
     */
    public static void main(String[] args)
    {
        boolean encrypt = true;
        String infile = null;
        String outfile = null;
        String keylen = null;

        if (args.length < 2)
        {
                  prRC4 rc = new prRC4();
            System.err.println("Usage: java "+rc.getClass().getName()+
                                " infile outfile keylen");
            System.exit(1);
        }
            infile = args[0];
            outfile = args[1];
            keylen = args[2];

            if (args.length > 3)
            {
                encrypt = false;
        }

        prRC4 rc = new prRC4(infile, outfile, keylen, encrypt);
        rc.process();
    }

    // Default constructor, used for the usage message
    public prRC4()
    {
    }
```

```
    /*
     * Constructor, that takes the arguments appropriate for
     * processing the command line directives.
     */
    public prRC4(
                String infile,
                String outfile,
                String keylen,
                boolean encrypt)
    {
            int i;
            String strKey = "";
        /*
         * First, determine that infile & keyfile exist as appropriate.
         *
         * This will also create the BufferedInputStream as required
         * for reading the input file. All input files are treated
         * as if they are binary, even if they contain text, it's the
         * bytes that are encrypted.
         */
        this.encrypt = encrypt;
        try
        {
            in = new BufferedInputStream(new FileInputStream(infile));
        }
        catch (FileNotFoundException fnf)
        {
            System.err.println("Input file not found ["+infile+"]");
            System.exit(1);
        }

        try
        {
            out = new BufferedOutputStream(new
FileOutputStream(outfile));
        }
        catch (IOException fnf)
        {
            System.err.println("Output file not created
["+outfile+"]");
            System.exit(1);
        }

            for (i =0;i < (Integer.parseInt(keylen));i++)
                  strKey += "0123456789abcdef";

            key = Hex.decode(strKey);
    }

    public final void process()
    {
        /*
         * Setup the DESede cipher engine, create a
PaddedBufferedBlockCipher
         * in CBC mode.
         */
        cipher = new RC4Engine();
```

```
    /*
     * The input and output streams are currently set up
     * appropriately, and the key bytes are ready to be
     * used.
     *
     */

    if (encrypt)
    {
        performEncrypt(key);
    }
    else
    {
        performDecrypt(key);
    }

    // after processing clean up the files
    try
    {
        in.close();
        out.flush();
        out.close();
    }
    catch (IOException closing)
    {

    }
}

/*
 * This method performs all the encryption and writes
 * the cipher text to the buffered output stream created
 * previously.
 */
private final void performEncrypt(byte[] key)
{
    // initialise the cipher with the key bytes, for encryption
    cipher.init(true, new KeyParameter(key));

    /*
     * Create some temporary byte arrays for use in
     * encryption, make them a reasonable size so that
     * we don't spend forever reading small chunks from
     * a file.
     *
     */
    // int inBlockSize = cipher.getBlockSize() * 5;
    int inBlockSize = 47;

    byte[] inblock = new byte[inBlockSize];
    byte[] outblock = new byte[inBlockSize];

    /*
     * now, read the file, and output the chunks
     */
    try
```

```
        {
            int inL=0;
            int outL=0;
            byte[] rv = null;
            while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
            {
                cipher.processBytes(inblock, 0, inL, outblock, 0);
                /*
                 * Before we write anything out, we need to make sure
                 * that we've got something to write out.
                 */
                if (outblock.length > 0)
                {
                    rv = Hex.encode(outblock, 0, outblock.length);
                    out.write(rv, 0, rv.length);
                    out.write('\n');
                }
            }

        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

    /*
     * This method performs all the decryption and writes
     * the plain text to the buffered output stream created
     * previously.
     */
    private final void performDecrypt(byte[] key)
    {
        // initialise the cipher for decryption
        cipher.init(false, new KeyParameter(key));

        /*
         * As the decryption is from our preformatted file,
         * and we know that it's a hex encoded format, then
         * we wrap the InputStream with a BufferedReader
         * so that we can read it easily.
         */
        BufferedReader br = new BufferedReader(new
InputStreamReader(in));

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int outL;
            byte[] inblock = null;
            byte[] outblock = null;
                  byte[]              input;
            String rv = null;
            while ((rv = br.readLine()) != null)
            {
```

```
                inblock = Hex.decode(rv);
                outblock = new byte[inblock.length];
                cipher.processBytes(inblock, 0, inblock.length,
outblock, 0);

                      if (outblock.length > 0)
                     out.write(outblock, 0, outblock.length);
            }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

}
```

```
* * * * * * * * * * * * * * * * * * * * * * *
*                                           *
*  Program name: prAES.java                 *
*                                           *
* * * * * * * * * * * * * * * * * * * * * * *
```

```
package unf.grad.proj;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.crypto.BufferedBlockCipher;
import org.bouncycastle.crypto.modes.CBCBlockCipher;
import org.bouncycastle.crypto.engines.AESEngine;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.util.encoders.Hex;

/**
 * @author Praveen Donta
 * File: prAES.java
 * Desc: encrypts and decrypts a given text file using AES algorithm.
 *
 */

public class prAES extends Object
{
    // Encrypting or decrypting ?
    private boolean encrypt = true;

    // To hold the initialised DESede cipher
    private BufferedBlockCipher cipher = null;
      //private OFBBlockCipher cipher = null;
```

```
// The input stream of bytes to be processed for encryption
private BufferedInputStream in = null;

// The output stream of bytes to be procssed
private BufferedOutputStream out = null;

// The key
private byte[] key = null;

/*
 * start the application
 */
public static void main(String[] args)
{
    boolean encrypt = true;
    String infile = null;
    String outfile = null;
    String keylen = null;

    if (args.length < 2)
    {
              prAES rc = new prAES();
        System.err.println("Usage: java "+rc.getClass().getName()+
                             " infile outfile");
        System.exit(1);
    }
        infile = args[0];
        outfile = args[1];
        keylen = args[2];

        if (args.length > 3)
        {
            encrypt = false;
    }

    prAES rc = new prAES(infile, outfile, keylen, encrypt);
    rc.process();
}

// Default constructor, used for the usage message
public prAES()
{
}

/*
 * Constructor, that takes the arguments appropriate for
 * processing the command line directives.
 */
public prAES(
            String infile,
            String outfile,
            String keylen,
            boolean encrypt)
{
        int i;
        String strKey = "";
```

```
        /*
         * First, determine that infile & keyfile exist as appropriate.
         *
         * This will also create the BufferedInputStream as required
         * for reading the input file. All input files are treated
         * as if they are binary, even if they contain text, it's the
         * bytes that are encrypted.
         */
        this.encrypt = encrypt;
        try
        {
            in = new BufferedInputStream(new FileInputStream(infile));
        }
        catch (FileNotFoundException fnf)
        {
            System.err.println("Input file not found ["+infile+"]");
            System.exit(1);
        }

        try
        {
            out = new BufferedOutputStream(new
FileOutputStream(outfile));
        }
        catch (IOException fnf)
        {
            System.err.println("Output file not created
["+outfile+"]");
            System.exit(1);
        }

            for (i =0;i < (Integer.parseInt(keylen));i++)
                  strKey += "0123456789abcdef";

            key = Hex.decode(strKey);

    }

    public final void process()
    {

        cipher = new PaddedBufferedBlockCipher(new CBCBlockCipher(new
AESEngine()));
        //cipher = new OFBBlockCipher(new AESEngine(), 16);

        /*
         * The input and output streams are currently set up
         * appropriately, and the key bytes are ready to be
         * used.
         *
         */

        if (encrypt)
        {
            performEncrypt(key);
        }
        else
```

```
        {
            performDecrypt(key);
        }

        // after processing clean up the files
        try
        {
            in.close();
            out.flush();
            out.close();
        }
        catch (IOException closing)
        {

        }
    }

    /*
     * This method performs all the encryption and writes
     * the cipher text to the buffered output stream created
     * previously.
     */
    private final void performEncrypt(byte[] key)
    {
        // initialise the cipher with the key bytes, for encryption
        cipher.init(true, new KeyParameter(key));

        /*
         * Create some temporary byte arrays for use in
         * encryption, make them a reasonable size so that
         * we don't spend forever reading small chunks from
         * a file.
         *
         */
        // int inBlockSize = cipher.getBlockSize() * 5;
        int inBlockSize = 47;
            int outBlockSize = cipher.getOutputSize(inBlockSize);

        byte[] inblock = new byte[inBlockSize];
        byte[] outblock = new byte[outBlockSize];

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int inL=0;
            int outL=0;
            byte[] rv = null;
            while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
            {
                outL = cipher.processBytes(inblock, 0, inL, outblock,
0);
               // cipher.processBlock(inblock, 0, outblock, 0);
                /*
                 * Before we write anything out, we need to make sure
                 * that we've got something to write out.
```

```
                 */
                if (outL > 0)
                {
                    rv = Hex.encode(outblock, 0, outL);
                    out.write(rv, 0, rv.length);
                    out.write('\n');
                }
            }

                  try
                  {
                        /*
                         * Now, process the bytes that are still
buffered
                         * within the cipher.
                         */
                        outL = cipher.doFinal(outblock, 0);
                        if (outL > 0)
                        {
                              rv = Hex.encode(outblock, 0, outL);
                              out.write(rv, 0, rv.length);
                              out.write('\n');
                        }
                  }
                  catch (CryptoException ce)
                  {
                  }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }


    }

    /*
     * This method performs all the decryption and writes
     * the plain text to the buffered output stream created
     * previously.
     */
    private final void performDecrypt(byte[] key)
    {
        // initialise the cipher for decryption
        cipher.init(false, new KeyParameter(key));

        /*
         * As the decryption is from our preformatted file,
         * and we know that it's a hex encoded format, then
         * we wrap the InputStream with a BufferedReader
         * so that we can read it easily.
         */
        BufferedReader br = new BufferedReader(new
InputStreamReader(in));

        /*
         * now, read the file, and output the chunks
```

```
         */
        try
        {
            int inL, outL;
          //  int inBlockSize = 16;
            byte[] inblock = null;
            byte[] outblock = null;
                  byte[]              input;
            String rv = null;
            while ((rv = br.readLine()) != null)
      //          while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
            {
               inblock = Hex.decode(rv);
               outblock = new
byte[cipher.getOutputSize(inblock.length)];
                     //outblock = new byte[inblock.length];
               outL = cipher.processBytes(inblock, 0, inblock.length,
outblock, 0);
               //cipher.processBlock(inblock, 0, outblock, 0);

                     if (outL > 0)
                    out.write(outblock, 0, outL);
            }

                  try
                  {
                        /*
                         * Now, process the bytes that are still
buffered
                         * within the cipher.
                         */
                        outL = cipher.doFinal(outblock, 0);
                        if (outL > 0)
                        {
                              out.write(outblock, 0, outL);
                        }
                  }
                  catch (CryptoException ce)
                  {

                  }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

}
```

*** ***
***   ***
*** Program name: prRSA.java   ***
***   ***
*** ***

```
/**
```

```
 * @author Praveen Donta
 * File: prRSA.java
 * Desc: encrypts and decrypts a given text file using RSA algorithm.
 *
 */

package unf.grad.proj;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.SecureRandom;

import org.bouncycastle.util.encoders.Hex;
import org.bouncycastle.crypto.engines.RSAEngine;
import org.bouncycastle.crypto.AsymmetricBlockCipher;
import org.bouncycastle.crypto.params.RSAKeyParameters;
import org.bouncycastle.crypto.generators.RSAKeyPairGenerator;
import org.bouncycastle.crypto.params.RSAPrivateCrtKeyParameters;
import org.bouncycastle.crypto.params.RSAKeyGenerationParameters;
import org.bouncycastle.crypto.AsymmetricCipherKeyPair;


public class prRSA extends Object
{
    // Encrypting or decrypting ?
    private boolean encrypt = true;

    // To hold the initialised DESede cipher
    private AsymmetricBlockCipher cipher = null;

    // The input stream of bytes to be processed for encryption
    private BufferedInputStream in = null;

    // The output stream of bytes to be procssed
    private BufferedOutputStream out = null;


    /*
     * start the application
     */
    public static void main(String[] args)
    {
        boolean encrypt = true;
        String infile = null;
        String outfile = null;
        String keylen = null;

        if (args.length < 3)
        {
```

```
                prRSA rsa = new prRSA();
            System.err.println("Usage: java "+rsa.getClass().getName()+
                                    " infile outfile keylength");
            System.exit(1);
        }
            infile = args[0];
            outfile = args[1];
            keylen = args[2];

            if (args.length > 3)
            {
                encrypt = false;
        }

        prRSA rsa = new prRSA(infile, outfile, encrypt);
        //rsa.process("768");
        //rsa.process(keylen);
        rsa.generateKey(keylen);
    }

    // Default constructor, used for the usage message
    public prRSA()
    {
    }

    /*
     * Constructor, that takes the arguments appropriate for
     * processing the command line directives.
     */
    public prRSA(
                String infile,
                String outfile,
                boolean encrypt)
    {
        /*
         * First, determine that infile & keyfile exist as appropriate.
         *
         * This will also create the BufferedInputStream as required
         * for reading the input file. All input files are treated
         * as if they are binary, even if they contain text, it's the
         * bytes that are encrypted.
         */
        this.encrypt = encrypt;
        try
        {
            in = new BufferedInputStream(new FileInputStream(infile));
        }
        catch (FileNotFoundException fnf)
        {
            System.err.println("Input file not found ["+infile+"]");
            System.exit(1);
        }

        try
        {
            out = new BufferedOutputStream(new
FileOutputStream(outfile));
```

```
        }
        catch (IOException fnf)
        {
            System.err.println("Output file not created
["+outfile+"]");
            System.exit(1);
        }

    }

    public final void process(String keylen)
    {
        cipher = new RSAEngine();

        /*
         * The input and output streams are currently set up
         * appropriately, and the key bytes are ready to be
         * used.
         *
         */

        if (encrypt)
        {
            performEncrypt(keylen);
        }
        else
        {
            performDecrypt();
        }

        // after processing clean up the files
        try
        {
            in.close();
            out.flush();
            out.close();
        }
        catch (IOException closing)
        {

        }
    }

    private final void generateKey(String keyLen)
    {
            String keyfile = "RSAkey.dat";

            RSAKeyPairGenerator  pGen = new RSAKeyPairGenerator();
            RSAKeyGenerationParameters  genParam = new
RSAKeyGenerationParameters(BigInteger.valueOf(0x11), new
SecureRandom(), Integer.parseInt(keyLen), 25);
            pGen.init(genParam);
            AsymmetricCipherKeyPair pair = pGen.generateKeyPair();

            BigInteger para;
            byte[] keyhex;
            try {
```

```
                    BufferedOutputStream keystream =
                          new BufferedOutputStream(new
FileOutputStream(keyfile));
                    keyhex =
((RSAKeyParameters)pair.getPublic()).getModulus().toString().getBytes()
;
                    keystream.write(keyhex, 0, keyhex.length);
                    keystream.write('\n');
                    keyhex =
((RSAKeyParameters)pair.getPublic()).getExponent().toString().getBytes(
);
                    keystream.write(keyhex, 0, keyhex.length);
                    keystream.write('\n');
                    keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getPublicExponent().toS
tring().getBytes();
                    keystream.write(keyhex, 0, keyhex.length);
                    keystream.write('\n');
                    keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getP().toString().getBy
tes();
                    keystream.write(keyhex, 0, keyhex.length);
                    keystream.write('\n');
                    keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getQ().toString().getBy
tes();
                    keystream.write(keyhex, 0, keyhex.length);
                    keystream.write('\n');
                    keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getDP().toString().getB
ytes();
                    keystream.write(keyhex, 0, keyhex.length);
                    keystream.write('\n');
                    keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getDQ().toString().getB
ytes();
                    keystream.write(keyhex, 0, keyhex.length);
                    keystream.write('\n');
                    keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getQInv().toString().ge
tBytes();
                    keystream.write(keyhex, 0, keyhex.length);

                    keystream.flush();
                    keystream.close();

              } catch (IOException createKey)
              {
                    System.err.println("Could not create key file "+

              "["+keyfile+"]");
                    System.exit(1);
              }
        }

     /*
      * This method performs all the encryption and writes
```

```
     * the cipher text to the buffered output stream created
     * previously.
     */
    private final void performEncrypt(String keyLen)
    {
      String srKeyfile="";
      if (keyLen.equals("512"))
            srKeyfile = "RSAkey512.dat";
      else if (keyLen.equals("768"))
            srKeyfile = "RSAkey768.dat";
      else if (keyLen.equals("1024"))
            srKeyfile = "RSAkey1024.dat";
      else if (keyLen.equals("2048"))
            srKeyfile = "RSAkey2048.dat";
            else if (keyLen.equals("3072"))
                  srKeyfile = "RSAkey3072.dat";
            else if (keyLen.equals("4096"))
                  srKeyfile = "RSAkey4096.dat";
            else if (keyLen.equals("5120"))
                  srKeyfile = "RSAkey5120.dat";

            String keyfile = "RSAkey.dat";
            /*
            RSAKeyPairGenerator  pGen = new RSAKeyPairGenerator();
            RSAKeyGenerationParameters  genParam = new
RSAKeyGenerationParameters(BigInteger.valueOf(0x11), new
SecureRandom(), Integer.parseInt(keyLen), 25);
            pGen.init(genParam);
            AsymmetricCipherKeyPair pair = pGen.generateKeyPair();

            BigInteger para;
            byte[] keyhex;
            try {
                  BufferedOutputStream keystream =
                        new BufferedOutputStream(new
FileOutputStream(keyfile));
                  keyhex =
((RSAKeyParameters)pair.getPublic()).getModulus().toString().getBytes()
;
                  keystream.write(keyhex, 0, keyhex.length);
                  keystream.write('\n');
                  keyhex =
((RSAKeyParameters)pair.getPublic()).getExponent().toString().getBytes(
);
                  keystream.write(keyhex, 0, keyhex.length);
                  keystream.write('\n');
                  keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getPublicExponent().toS
tring().getBytes();
                  keystream.write(keyhex, 0, keyhex.length);
                  keystream.write('\n');
                  keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getP().toString().getBy
tes();
                  keystream.write(keyhex, 0, keyhex.length);
                  keystream.write('\n');
```

```
                  keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getQ().toString().getBy
tes();
                  keystream.write(keyhex, 0, keyhex.length);
                  keystream.write('\n');
                  keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getDP().toString().getB
ytes();
                  keystream.write(keyhex, 0, keyhex.length);
                  keystream.write('\n');
                  keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getDQ().toString().getB
ytes();
                  keystream.write(keyhex, 0, keyhex.length);
                  keystream.write('\n');
                  keyhex =
((RSAPrivateCrtKeyParameters)pair.getPrivate()).getQInv().toString().ge
tBytes();
                  keystream.write(keyhex, 0, keyhex.length);

                  keystream.flush();
                  keystream.close();

            } catch (IOException createKey)
            {
                  System.err.println("Could not create key file "+

            "["+keyfile+"]");
                  System.exit(1);
            }
      */

        // initialise the cipher with the key bytes, for encryption
            //cipher.init(true, pair.getPublic());

            BigInteger mod=null, pubExp=null;
            String bv;
            int i;

            try {
                  BufferedInputStream keystream =
                        new BufferedInputStream(new
FileInputStream(srKeyfile));
                  BufferedReader keyReader = new BufferedReader(new
InputStreamReader(keystream));
                  PrintWriter pr = new PrintWriter(new
FileOutputStream(keyfile));

                  // write variables mod, pubExp, privExp, p, q, dp,
dq, qinv
                  if ((bv = keyReader.readLine()) != null)
                         pr.println(bv);
                  mod = new BigInteger(bv);
                  // pubExp
                  if ((bv = keyReader.readLine()) != null)
                         pr.println(bv);
                  pubExp = new BigInteger(bv);
```

```
                  for (i=0; i < 6;i++)
                        if ((bv = keyReader.readLine()) != null)
                              pr.println(bv);

                  keystream.close();
                  keyReader.close();
                  pr.close();

            } catch (IOException createKey)     {
                  System.err.println("Could not encyption create key
file "+ "["+keyfile+"]");
                  System.exit(1);
            }

            RSAKeyParameters    pubParameters = new
RSAKeyParameters(false, mod, pubExp);
            cipher.init(true, pubParameters);

        /*
         * Create some temporary byte arrays for use in
         * encryption, make them a reasonable size so that
         * we don't spend forever reading small chunks from
         * a file.
         *
         * There is no particular reason for using getBlockSize()
         * to determine the size of the input chunk. It just
         * was a convenient number for the example.
         */
        // int inBlockSize = cipher.getBlockSize() * 5;
        int inBlockSize = 47;

        byte[] inblock = new byte[inBlockSize];
        byte[] outblock = new byte[inBlockSize];

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int inL=0;
            int outL=0;
            byte[] rv = null;
            String st = null;
                  BufferedReader br = new BufferedReader(new
InputStreamReader(in));
           // while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
                  while ((st = br.readLine()) != null)
            {
                        inblock = st.getBytes();
                  try {
                              outblock = cipher.processBlock(inblock,
0, inblock.length);

                  /*
                  * Before we write anything out, we need to make sure
                  * that we've got something to write out.
```

```
                  */
                  if (outblock.length > 0)
                  {
                        rv = Hex.encode(outblock, 0, outblock.length);
                        out.write(rv, 0, rv.length);
                        out.write('\n');
                      }
                  } catch (Exception e)
                        {
                              System.err.println("failed - exception "
+ e.toString());
                        }
            }

            in.close();
                  out.flush();
                  out.close();
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

    /*
     * This method performs all the decryption and writes
     * the plain text to the buffered output stream created
     * previously.
     */
    private final void performDecrypt()
    {

            String bv;
            BigInteger mod=null, pubExp=null, privExp=null, p=null,
q=null, dp=null, dq=null, qinv=null;
            String keyfile = "ServRSAKey.dat";
            try {
                        BufferedInputStream keystream =
                              new BufferedInputStream(new
FileInputStream(keyfile));
                        BufferedReader keyReader = new
BufferedReader(new InputStreamReader(keystream));
                        if ((bv = keyReader.readLine()) != null)
                        {
                              mod = new BigInteger(bv);
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              pubExp = new BigInteger(bv);
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              privExp = new BigInteger(bv);
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              p = new BigInteger(bv);
```

```
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              q = new BigInteger(bv);
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              dp = new BigInteger(bv);
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              dq = new BigInteger(bv);
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              qinv = new BigInteger(bv);
                        }

                        keystream.close();
                        keyReader.close();

                  } catch (IOException createKey)
                  {
                        System.err.println("Could not decryption create
key file "+

      "["+keyfile+"]");
                                                System.exit(1);
                  }


        // initialise the cipher for decryption
            RSAKeyParameters    privParameters = new
RSAPrivateCrtKeyParameters(mod, pubExp, privExp, p, q, dp, dq, qinv);
            cipher.init(false, privParameters);

        /*
         * As the decryption is from our preformatted file,
         * and we know that it's a hex encoded format, then
         * we wrap the InputStream with a BufferedReader
         * so that we can read it easily.
         */
        BufferedReader br = new BufferedReader(new
InputStreamReader(in));

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int outL;
            byte[] inblock = null;
            byte[] outblock = null;
                  byte[]              input;
            String rv = null;
            while ((rv = br.readLine()) != null)
            {
```

```
                inblock = Hex.decode(rv);
                outblock = new byte[inblock.length];
                try {
                            outblock = cipher.processBlock(inblock, 0,
inblock.length);

                            String te = new String(outblock);
                            if (outblock.length > 0) {
                           out.write(outblock, 0, outblock.length);
                           out.write('\n');
                            }

                } catch (Exception e) {
                               System.err.println("failed - exception "
+ e.toString());
                }
             }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

}
```

*** ***
***   ***
*** Program name: prElGamal.java   ***
***   ***
*** ***

```
/**
 * @author Praveen Donta
 * File: prElGamal.java
 * Desc: encrypts and decrypts a given text file using ElGamal
algorithm.
 *
 */

package unf.grad.proj;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.SecureRandom;

import org.bouncycastle.crypto.engines.ElGamalEngine;
import org.bouncycastle.crypto.AsymmetricCipherKeyPair;
import org.bouncycastle.crypto.generators.ElGamalKeyPairGenerator;
```

```
import org.bouncycastle.crypto.generators.ElGamalParametersGenerator;
import org.bouncycastle.crypto.params.ElGamalParameters;
import org.bouncycastle.crypto.params.ElGamalKeyGenerationParameters;
import org.bouncycastle.crypto.params.ElGamalPrivateKeyParameters;
import org.bouncycastle.crypto.params.ElGamalPublicKeyParameters;
import org.bouncycastle.crypto.params.ParametersWithRandom;
import org.bouncycastle.util.encoders.Hex;

public class prElGamal extends Object
{
    // Encrypting or decrypting ?
    private boolean encrypt = true;

    // To hold the initialised DESede cipher
    private ElGamalEngine    e = null;

    // The input stream of bytes to be processed for encryption
    private BufferedInputStream in = null;

    // The output stream of bytes to be procssed
    private BufferedOutputStream out = null;

    /*
     * start the application
     */
    public static void main(String[] args)
    {
        boolean encrypt = true;
        String infile = null;
        String outfile = null;
        String keylen = null;

        if (args.length < 3)
        {
                  prElGamal rc = new prElGamal();
            System.err.println("Usage: java "+rc.getClass().getName()+
                                    " infile outfile keylength");
            System.exit(1);
        }

            infile = args[0];
            outfile = args[1];
            keylen = args[2];

            if (args.length > 3)
            {
                encrypt = false;
        }

        prElGamal el = new prElGamal(infile, outfile, encrypt);
        //el.process("258");
        el.process(keylen);
    }

    // Default constructor, used for the usage message
    public prElGamal()
    {
```

```
    }

    /*
     * Constructor, that takes the arguments appropriate for
     * processing the command line directives.
     */
    public prElGamal(
                String infile,
                String outfile,
                boolean encrypt)
    {
        /*
         * First, determine that infile & keyfile exist as appropriate.
         *
         * This will also create the BufferedInputStream as required
         * for reading the input file. All input files are treated
         * as if they are binary, even if they contain text, it's the
         * bytes that are encrypted.
         */
        this.encrypt = encrypt;
        try
        {
            in = new BufferedInputStream(new FileInputStream(infile));
        }
        catch (FileNotFoundException fnf)
        {
            System.err.println("Input file not found ["+infile+"]");
            System.exit(1);
        }

        try
        {
            out = new BufferedOutputStream(new
FileOutputStream(outfile));
        }
        catch (IOException fnf)
        {
            System.err.println("Output file not created
["+outfile+"]");
            System.exit(1);
        }
    }

    public final void process(String keylen)
    {
        e = new ElGamalEngine();

            // testGeneration(258);

        if (encrypt)
        {
            performEncrypt(keylen);
        }
        else
        {
            performDecrypt();
        }
```

```
        // after processing clean up the files
        try
        {
            in.close();
            out.flush();
            out.close();
        }
        catch (IOException closing)
        {

        }
    }

      private void testGeneration(
               int          size)
         {
               ElGamalParametersGenerator       pGen = new
ElGamalParametersGenerator();

               pGen.init(size, 10, new SecureRandom());

               ElGamalParameters                elParams =
pGen.generateParameters();

               ElGamalKeyGenerationParameters   params = new
ElGamalKeyGenerationParameters(new SecureRandom(), elParams);

               ElGamalKeyPairGenerator          kpGen = new
ElGamalKeyPairGenerator();

               kpGen.init(params);

               //
               // generate first pair
               //
               AsymmetricCipherKeyPair         pair =
kpGen.generateKeyPair();

               ElGamalPublicKeyParameters      pu =
(ElGamalPublicKeyParameters)pair.getPublic();
               ElGamalPrivateKeyParameters     pv =
(ElGamalPrivateKeyParameters)pair.getPrivate();

               ElGamalEngine    e = new ElGamalEngine();

               e.init(true, new ParametersWithRandom(pu, new
SecureRandom()));

               String  message = "This is a test";

               byte[]  pText = message.getBytes();
               byte[]  cText = e.processBlock(pText, 0, pText.length);

               e.init(false, pv);

               pText = e.processBlock(cText, 0, cText.length);
```

```
                if (!message.equals(new String(pText)))
                {
                      System.out.println("generation test failed");
                }
          }

     /*
      * This method performs all the encryption and writes
      * the cipher text to the buffered output stream created
      * previously.
      */
     private final void performEncrypt(String keylen)
     {
       /*
            ElGamalParametersGenerator       pGen = new
ElGamalParametersGenerator();

            //pGen.init(258, 10, new SecureRandom());
            pGen.init(Integer.parseInt(keylen), 10, new
SecureRandom());

            ElGamalParameters                elParams =
pGen.generateParameters();
            */

            String srKeyfile="";
            String bv;
            BigInteger p=null, g=null;

            if (keylen.equals("600"))
                  srKeyfile = "ElGamalKey600.dat";
            else if (keylen.equals("800"))
                  srKeyfile = "ElGamalKey800.dat";
            else if (keylen.equals("1000"))
                  srKeyfile = "ElGamalKey1000.dat";
            else if (keylen.equals("1200"))
                  srKeyfile = "ElGamalKey1200.dat";

            String keyfile = "ElGamalKey.dat";

            try {
                  BufferedInputStream keystream =
                        new BufferedInputStream(new
FileInputStream(srKeyfile));
                  BufferedReader keyReader = new BufferedReader(new
InputStreamReader(keystream));
                  PrintWriter pr = new PrintWriter(new
FileOutputStream(keyfile));

                  // write variables p, g, x
                  if ((bv = keyReader.readLine()) != null)
                         pr.println(bv);
                  p = new BigInteger(bv);

                  if ((bv = keyReader.readLine()) != null)
                         pr.println(bv);
```

```
                g = new BigInteger(bv);

                if ((bv = keyReader.readLine()) != null)
                      pr.println(bv);

                keystream.close();
                keyReader.close();
                pr.close();

          } catch (IOException createKey)     {
                System.err.println("Could not decryption create key
file "+ "["+keyfile+"]");
                System.exit(1);
          }

          ElGamalParameters                elParams = new
ElGamalParameters(p, g);
          ElGamalKeyGenerationParameters   params = new
ElGamalKeyGenerationParameters(new SecureRandom(), elParams);
          ElGamalKeyPairGenerator          kpGen = new
ElGamalKeyPairGenerator();
          kpGen.init(params);

          //
          // generate first pair
          //
          AsymmetricCipherKeyPair         pair =
kpGen.generateKeyPair();

          ElGamalPublicKeyParameters      pu =
(ElGamalPublicKeyParameters)pair.getPublic();
          ElGamalPrivateKeyParameters     pv =
(ElGamalPrivateKeyParameters)pair.getPrivate();

          // Store Key
          byte[] keyhex;
          try {
                BufferedOutputStream keystream =
                      new BufferedOutputStream(new
FileOutputStream(keyfile));
                keyhex = elParams.getP().toString().getBytes();
                keystream.write(keyhex, 0, keyhex.length);
                keystream.write('\n');
                keyhex = elParams.getG().toString().getBytes();
                keystream.write(keyhex, 0, keyhex.length);
                keystream.write('\n');
                keyhex = pv.getX().toString().getBytes();
                keystream.write(keyhex, 0, keyhex.length);
                keystream.write('\n');

                keystream.flush();
                keystream.close();

          } catch (IOException createKey)
          {
                System.err.println("Could not decryption create key
file "+
```

```
            "["+keyfile+"]");
                  System.exit(1);
            }


        e.init(true, new ParametersWithRandom(pu, new SecureRandom()));

        // int inBlockSize = cipher.getBlockSize() * 5;
        int inBlockSize = 47;

        byte[] inblock = new byte[inBlockSize];
        byte[] outblock = new byte[inBlockSize];

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int inL=0;
             byte[] rv = null;
            byte[] cText;

            while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
            {
                        cText = e.processBlock(inblock, 0, inL);

                if (cText.length > 0)
                {
                    rv = Hex.encode(cText, 0, cText.length);
                    out.write(rv, 0, rv.length);
                    out.write('\n');
                }
            }

        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

    /*
     * This method performs all the decryption and writes
     * the plain text to the buffered output stream created
     * previously.
     */
    private final void performDecrypt()
    {

            String bv;
            BigInteger p=null, g=null, x=null;
            String keyfile = "ServElGamalKey.dat";
            try {
                        BufferedInputStream keystream =
                              new BufferedInputStream(new
FileInputStream(keyfile));
```

```
                        BufferedReader keyReader = new
BufferedReader(new InputStreamReader(keystream));

                        if ((bv = keyReader.readLine()) != null)
                        {
                              p = new BigInteger(bv);
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              g = new BigInteger(bv);
                        }
                        if ((bv = keyReader.readLine()) != null)
                        {
                              x = new BigInteger(bv);
                        }

                        keystream.close();
                        keyReader.close();

            } catch (IOException createKey)
            {
                        System.err.println("Could not decryption create
key file "+

            "["+keyfile+"]");
                        System.exit(1);
            }

            ElGamalParameters               elParams = new
ElGamalParameters(p, g);
            ElGamalPrivateKeyParameters     pv = new
ElGamalPrivateKeyParameters(x, elParams);

        // initialise the cipher for decryption
        e.init(false, pv);

        /*
         * As the decryption is from our preformatted file,
         * and we know that it's a hex encoded format, then
         * we wrap the InputStream with a BufferedReader
         * so that we can read it easily.
         */
        BufferedReader br = new BufferedReader(new
InputStreamReader(in));

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            byte[] inblock = null;
            byte[] outblock = null;
            byte[] pText;
            String rv = null;
            while ((rv = br.readLine()) != null)
            {
               inblock = Hex.decode(rv);
```

```
                outblock = new byte[inblock.length];
                pText = e.processBlock(inblock, 0, inblock.length);

                      if (pText.length > 0)
                     out.write(pText, 0, pText.length);
            }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }
    }

}
```

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                 *
*  Program name: prSHA1.java                                      *
*                                                                 *
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
```

```
/**
 * @author Praveen Donta
 * File: prSHA1.java
 * Desc: generates message digest for a given text file using SHA1.
 *
 */

package unf.grad.proj;

import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import org.bouncycastle.util.encoders.Hex;
import org.bouncycastle.crypto.digests.SHA1Digest;

public class prSHA1 extends Object
{
      private SHA1Digest digest;

    // The input stream of bytes to be processed for digest
    private BufferedInputStream in = null;

    /*
     * start the application
     */
    public static void main(String[] args)
    {
        String infile = null;

        if (args.length < 2)
        {
            prSHA1 de = new prSHA1();
            System.err.println("Usage: java "+de.getClass().getName()+
```

```
                        " infile outfile");
        System.exit(1);
    }
        infile = args[0];

    prSHA1 pr = new prSHA1(infile);
    pr.process();
}

// Default constructor, used for the usage message
public prSHA1()
{
}

/*
 * Constructor, that takes the arguments appropriate for
 * processing the command line directives.
 */
public prSHA1(String infile)
{
    try
    {
        in = new BufferedInputStream(new FileInputStream(infile));
    }
    catch (FileNotFoundException fnf)
    {
        System.err.println("Input file not found ["+infile+"]");
        System.exit(1);
    }
}

public final byte[] process()
{
  byte[] rv;
    /*
     * Create Digest
     *
     */
    digest  = new SHA1Digest();

    /*
     * The input and output streams are currently set up
     * appropriately, and the key bytes are ready to be
     * used.
     *
     */

    rv = performDigest();

    // after processing clean up the files
    try
    {
        in.close();
    }
    catch (IOException closing)
    {
```

```
        }

        return rv;
    }

    /*
     * This method generates digest
     *
     *
     */
    private final byte[] performDigest()
    {
        int inBlockSize = 47;

        byte[] inblock = new byte[inBlockSize];

        /*
         * now, read the file, and output the chunks
         */
        try
        {
            int inL;
                  byte[] resBuf = new byte[digest.getDigestSize()];
                  byte rv[];
            while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
            {
                digest.update(inblock, 0, inL);
            }
                  digest.doFinal(resBuf, 0);
                  if (resBuf.length > 0)
                  {
                        rv = Hex.encode(resBuf, 0, resBuf.length);
                        return rv;
                  }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }

        return null;
    }
}
```

```
* * * * * * * * * * * * * * * * * * * * * * *
*                                           *
*  Program name: prMD5.java                 *
*                                           *
* * * * * * * * * * * * * * * * * * * * * * *
```

```
/**
 * @author Praveen Donta
 * File: prMD5.java
 * Desc: generates message digest for a given text file using MD5.
 *
 */
```

```
package unf.grad.proj;

import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import org.bouncycastle.util.encoders.Hex;
import org.bouncycastle.crypto.digests.MD5Digest;

public class prMD5 extends Object
{
      private MD5Digest digest;

    // The input stream of bytes to be processed for encryption
    private BufferedInputStream in = null;

    /*
     * start the application
     */
    public static void main(String[] args)
    {
        boolean encrypt = true;
        String infile = null;

        if (args.length < 2)
        {
            prMD5 de = new prMD5();
            System.err.println("Usage: java "+de.getClass().getName()+
                                " infile outfile");
            System.exit(1);
        }
            infile = args[0];

        prMD5 pr = new prMD5(infile);
        pr.process();
    }

    // Default constructor, used for the usage message
    public prMD5()
    {
    }

    /*
     * Constructor, that takes the arguments appropriate for
     * processing the command line directives.
     */
    public prMD5( String infile)
    {
        try
        {
            in = new BufferedInputStream(new FileInputStream(infile));
        }
        catch (FileNotFoundException fnf)
        {
            System.err.println("Input file not found ["+infile+"]");
```

```
        System.exit(1);
    }
}

public final byte[] process()
{
  byte[] rv;
    digest  = new MD5Digest();

    /*
     * The input and output streams are currently set up
     * appropriately, and the key bytes are ready to be
     * used.
     *
     */

    rv = performDigest();

    // after processing clean up the files
    try
    {
        in.close();
    }
    catch (IOException closing)
    {

    }

    return rv;
}

/*
 * This method performs digest
 */
private final byte[] performDigest()
{
    int inBlockSize = 47;

    byte[] inblock = new byte[inBlockSize];

        byte[] rv;

    /*
     * now, read the file, and output the chunks
     */
    try
    {
        int inL;
              byte[] resBuf = new byte[digest.getDigestSize()];
        while ((inL=in.read(inblock, 0, inBlockSize)) > 0)
        {
            digest.update(inblock, 0, inL);
        }
              digest.doFinal(resBuf, 0);
              if (resBuf.length > 0)
              {
                    rv = Hex.encode(resBuf, 0 , resBuf.length);
```

```
                    return rv;
                }
        }
        catch (IOException ioeread)
        {
            ioeread.printStackTrace();
        }

        return null;
    }
}
```

Appendix B

SSL CODE LISTINGS

```
* * * * * * * * * * * * * * * * * * * * * * *
*                                           *
* Program name: SSLSecurityServer.cpp       *
*                                           *
* * * * * * * * * * * * * * * * * * * * * * *

/*
 * SecurityServer.cpp
 * Author: Praveen Donta
 * Desc: SSL security server handles SSL client connections
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <memory.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/timeb.h>
#include <sys/time.h>

#include <openssl/rsa.h>       /* SSLeay stuff */
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/err.h>


/* define HOME to be dir for key and cert files... */
#define HOME "./"

#define CERTF  HOME "RSACert2048.pem"
#define KEYF  HOME  "RSACert2048.pem"


#define CHK_NULL(x) if ((x)==NULL) exit (1)
#define CHK_ERR(err,s) if ((err)==-1) { perror(s); exit(1); }
#define CHK_SSL(err) if ((err)==-1) { ERR_print_errors_fp(stderr);
exit(2); }

int processInputchoice(char *, int);
int processRequest(int);
```

```
int main (int argc, char argv[])
{
  char hostName[10];
  char ret;

  // Check for number of command line arguments
  if (argc < 2) {
            printf("Missing server port. Please enter server port.");
            exit(0);
  }

  for (;;) {
            printf("\n\n");
            printf("1. 3DES-SHA with 512 bit RSA\n");
            printf("2. 3DES-SHA with 768 bit RSA\n");
            printf("3. 3DES-SHA with 1024 bit RSA\n");
            printf("4. 3DES-SHA with 2048 bit RSA\n");
            printf("5. RC4-MD5 with 512 bit RSA\n");
            printf("6. RC4-MD5 with 768 bit RSA\n");
            printf("7. RC4-MD5 with 1024 bit RSA\n");
            printf("8. RC4-MD5 with 2048 bit RSA\n");
            printf("9. Quit\n");
            printf("\n");
            printf("\nPlease Enter your choice (1-9) :");
          scanf("%s", &ret);
            fflush(stdin);
            if (ret == '9') {
                  exit(0);
            }
            processRequest(ret);
  }

}

int processInputchoice(char *hostName, int choice) {
      FILE *fp, *fout;
      char code[15];
      char outfile[15];
      int i, len;

      char inputfile[15];

    strcpy(inputfile, "TestFile.txt");

      //Open input file
      fp = fopen("TestFile", "r");
      if (fp == NULL) {
            printf("Cannot open input file\n");
            return 0;
      }

      len = strlen(inputfile);
      len -= 3;
      strncpy(outfile, inputfile, len);
      outfile[len] = 0;
      strcat(outfile, ".out");
```

```
        //Open output file
        fout = fopen(outfile, "w");
        if (fout == NULL) {
              printf("Cannot open output file\n");
              return 0;
        }

        printf("\n %d  Outfile %s \n", len, outfile);

        printf("\n Inputfile %s Successfully processed\n",inputfile);
        fclose(fp);
        fclose(fout);

        return 0;
}

int processRequest(int choice)
{
  int err;
  int listen_sd;
  int sd;
  struct sockaddr_in sa_serv;
  struct sockaddr_in sa_cli;
  //size_t client_len;
  socklen_t client_len;
  SSL_CTX* ctx;
  SSL*     ssl;
  X509*    client_cert;
  char*    str;
  char     buf [14096];
  char certFile[20];
  char keyFile[20];
  struct timeval start_time, stop_time;
  struct timezone szone;
  double elapsed;

  SSL_METHOD *meth;

  /* SSL preliminaries. We keep the certificate and key with the
context. */

  SSL_load_error_strings();
  SSL_library_init();
  meth=SSLv3_server_method();
  ctx = SSL_CTX_new (meth);
  if (choice == '1') {
      SSL_CTX_set_cipher_list(ctx,"DES-CBC3-SHA");
      strcpy(certFile, "RSACert512.pem");
      strcpy(keyFile, "RSACert512.pem");
  } else if (choice == '2') {
      SSL_CTX_set_cipher_list(ctx,"DES-CBC3-SHA");
      strcpy(certFile, "RSACert768.pem");
      strcpy(keyFile, "RSACert768.pem");
  } else if (choice == '3') {
      SSL_CTX_set_cipher_list(ctx,"DES-CBC3-SHA");
      strcpy(certFile, "RSACert1024.pem");
```

```
      strcpy(keyFile, "RSACert1024.pem");
  } else if (choice == '4') {
      SSL_CTX_set_cipher_list(ctx,"DES-CBC3-SHA");
      strcpy(certFile, "RSACert2048.pem");
      strcpy(keyFile, "RSACert2048.pem");
  } else if (choice == '5') {
      printf("In choice 5\n");
      SSL_CTX_set_cipher_list(ctx,"RC4-MD5");
      strcpy(certFile, "RSACert512.pem");
      strcpy(keyFile, "RSACert512.pem");
  } else if (choice == '6') {
      SSL_CTX_set_cipher_list(ctx,"RC4-MD5");
      strcpy(certFile, "RSACert768.pem");
      strcpy(keyFile, "RSACert768.pem");
  } else if (choice == '7') {
      SSL_CTX_set_cipher_list(ctx,"RC4-MD5");
      strcpy(certFile, "RSACert1024.pem");
      strcpy(keyFile, "RSACert1024.pem");
  } else if (choice == '8') {
      SSL_CTX_set_cipher_list(ctx,"RC4-MD5");
      strcpy(certFile, "RSACert2048.pem");
      strcpy(keyFile, "RSACert2048.pem");
  }

  printf("CTX Session key %d\n", ctx->generate_session_id);
  if (!ctx) {
    ERR_print_errors_fp(stderr);
    exit(2);
  }

  if (SSL_CTX_use_certificate_file(ctx, certFile, SSL_FILETYPE_PEM) <=
0) {
    ERR_print_errors_fp(stderr);
    exit(3);
  }

  if (SSL_CTX_use_PrivateKey_file(ctx, keyFile, SSL_FILETYPE_PEM) <= 0)
{
    ERR_print_errors_fp(stderr);
    exit(4);
  }

  if (!SSL_CTX_check_private_key(ctx)) {
    fprintf(stderr,"Private key does not match the certificate public
key\n");
    exit(5);
  }

  /* ----------------------------------------------- */
  /* Prepare TCP socket for receiving connections */

  listen_sd = socket (AF_INET, SOCK_STREAM, 0);   CHK_ERR(listen_sd,
"socket");

  memset (&sa_serv, '\0', sizeof(sa_serv));
  sa_serv.sin_family      = AF_INET;
  sa_serv.sin_addr.s_addr = INADDR_ANY;
```

```
  sa_serv.sin_port        = htons (1111);          /* Server Port
number */

  err = bind(listen_sd, (struct sockaddr*) &sa_serv,
           sizeof (sa_serv));                  CHK_ERR(err, "bind");

  /* Receive a TCP connection. */

  err = listen (listen_sd, 5);                    CHK_ERR(err,
"listen");

  client_len = sizeof(sa_cli);
  sd = accept (listen_sd, (struct sockaddr*) &sa_cli, &client_len);
  CHK_ERR(sd, "accept");
  close (listen_sd);

  printf ("Connection from %lx, port %x\n",
        sa_cli.sin_addr.s_addr, sa_cli.sin_port);

  /* ----------------------------------------------- */
  /* TCP connection is ready. Do server side SSL. */

    ssl = SSL_new (ctx);                           CHK_NULL(ssl);
    printf("SSL Session key before %d\n", ssl->session->session_id);

  SSL_set_fd (ssl, sd);
  printf("SSL Session key before %d\n", ssl->session->session_id);

  err = SSL_accept (ssl);                        CHK_SSL(err);
  printf("SSL Session key %d\n", ssl->session->session_id);

  /* Get the cipher - opt */

  printf ("SSL connection using %s\n",
SSL_CIPHER_get_name(SSL_get_current_cipher(ssl)));

  /* Get client's certificate (note: beware of dynamic allocation) -
opt */

  client_cert = SSL_get_peer_certificate (ssl);

  if (client_cert != NULL) {
    printf ("Client certificate:\n");

    str = X509_NAME_oneline (X509_get_subject_name (client_cert), 0,
0);
    CHK_NULL(str);
    printf ("\t subject: %s\n", str);
    OPENSSL_free (str);

    str = X509_NAME_oneline (X509_get_issuer_name  (client_cert), 0,
0);
    CHK_NULL(str);
    printf ("\t issuer: %s\n", str);
    OPENSSL_free (str);
```

```
    /* We could do all sorts of certificate verification stuff here
before
       deallocating the certificate. */

    X509_free (client_cert);
  } else
    printf ("Client does not have certificate.\n");

  /* DATA EXCHANGE - Receive message and send reply. */

  // Start timing
  gettimeofday(&start_time, &szone);
  printf("Start time %7f\n ", (double)start_time.tv_usec);

  err = 1;
 // while (err > 0) {
      err = SSL_read (ssl, buf, sizeof(buf) - 1);                 //
CHK_SSL(err);
      buf[err] = '\0';
      printf ("Got %d chars: \n", err);
 // }

  err = SSL_write (ssl, "I hear you.", strlen("I hear you."));
CHK_SSL(err);

  // Stop timing and compute elapsed time
  gettimeofday(&stop_time, &szone);
  printf("Stop time %f, %f \n ", (double)stop_time.tv_usec);

  elapsed=(((double) stop_time.tv_usec ) - ((double) start_time.tv_usec
));

  // Output name and elapsed time
  printf("  Server took %7.9f micro sec to execute this program \n",
elapsed);
  printf("-------------------------------------------------------------
--\n");

  /* Clean up. */

  close (sd);
  SSL_free (ssl);
  SSL_CTX_free (ctx);

  return 0;
}
/* EOF - SSLSecurityServer.cpp */
```

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                 *
*  Program name: SSLSecurityClient.cpp                            *
*                                                                 *
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
```

```
/*
 * SecurityServer.cpp
```

```
 * Author: Praveen Donta
 * Desc: SSL security client sends encrypted requests to SSL server
 *
 */

#include <stdio.h>
#include <memory.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/timeb.h>
#include <sys/time.h>

#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/err.h>


#define CHK_NULL(x) if ((x)==NULL) exit (1)
#define CHK_ERR(err,s) if ((err)==-1) { perror(s); exit(1); }
#define CHK_SSL(err) if ((err)==-1) { ERR_print_errors_fp(stderr);
exit(2); }

int main ()
{
  int err;
  int sd;
  struct sockaddr_in sa;
  SSL_CTX* ctx;
  SSL*     ssl;
  X509*    server_cert;
  char*    str;
  char     buf [14096];
  SSL_METHOD *meth;
  struct timeval start_time, stop_time;
  struct timezone szone;
  double elapsed;
  FILE *fp;

  SSLeay_add_ssl_algorithms();
  meth = SSLv3_client_method();
  SSL_load_error_strings();
  ctx = SSL_CTX_new (meth);                        CHK_NULL(ctx);

  CHK_SSL(err);

  /* ----------------------------------------------- */
  /* Create a socket and connect to server using normal socket calls.
*/

  sd = socket (AF_INET, SOCK_STREAM, 0);       CHK_ERR(sd, "socket");
```

```
  memset (&sa, '\0', sizeof(sa));
  sa.sin_family      = AF_INET;
  sa.sin_addr.s_addr = inet_addr ("127.0.0.1");   /* Server IP */
  sa.sin_port        = htons     (1111);          /* Server Port number
*/

  err = connect(sd, (struct sockaddr*) &sa,
            sizeof(sa));                   CHK_ERR(err, "connect");

  /* ----------------------------------------------- */
  /* Now we have TCP conncetion. Start SSL negotiation. */

  ssl = SSL_new (ctx);                         CHK_NULL(ssl);
  SSL_set_fd (ssl, sd);
  err = SSL_connect (ssl);                     CHK_SSL(err);

  /* Following two steps are optional and not required for
     data exchange to be successful. */

  /* Get the cipher - opt */

  printf ("SSL connection using %s\n",
SSL_CIPHER_get_name(SSL_get_current_cipher(ssl)));

  /* Get server's certificate (note: beware of dynamic allocation) -
opt */

  server_cert = SSL_get_peer_certificate (ssl);
CHK_NULL(server_cert);
  printf ("Server certificate:\n");

  str = X509_NAME_oneline (X509_get_subject_name (server_cert),0,0);
  CHK_NULL(str);
  printf ("\t subject: %s\n", str);
  OPENSSL_free (str);

  str = X509_NAME_oneline (X509_get_issuer_name  (server_cert),0,0);
  CHK_NULL(str);
  printf ("\t issuer: %s\n", str);
  OPENSSL_free (str);

  /* We could do all sorts of certificate verification stuff here
before
     deallocating the certificate. */

  X509_free (server_cert);

  /* --------------------------------------------------- */
  /* DATA EXCHANGE - Send a message and receive a reply. */

  fp = fopen("TestFile10k.txt", "r");

  // Start timing
  gettimeofday(&start_time, &szone);
  printf("Start time %7f\n ", (double)start_time.tv_usec);

  while (feof(fp) == 0) {
```

```
            fread(buf, sizeof(buf) -1, 1, fp);
            //printf("%s\n", buf);
            err = SSL_write(ssl,buf, sizeof(buf)); CHK_SSL(err);
            fread(buf, sizeof(buf) -1, 1, fp);
                        //printf("%s\n", buf);
            err = SSL_write(ssl,buf, sizeof(buf)); CHK_SSL(err);
  }

  err = SSL_write (ssl, "Hello World!", strlen("Hello World!"));
CHK_SSL(err);
  printf("Hello World\n");

  err = SSL_read (ssl, buf, sizeof(buf) - 1);
CHK_SSL(err);
  buf[err] = '\0';
  printf ("Got %d chars:'%s'\n", err, buf);

  // Stop timing and compute elapsed time
  gettimeofday(&stop_time, &szone);
  printf("Stop time %f\n ", (double)stop_time.tv_usec);

  elapsed=(((double) stop_time.tv_usec ) - ((double) start_time.tv_usec
));


  // Output name and elapsed time
  printf("  Client took %7.9f micro sec to execute this program \n",
elapsed);
  printf("-----------------------------------------------------------
--\n");

  SSL_shutdown (ssl);  /* send SSL/TLS close_notify */

  /* Clean up. */

//  close (sd);
  SSL_free (ssl);
  SSL_CTX_free (ctx);
}
/* EOF - SSLSecurityClient.cpp */
```

VITA

Praveen K. Donta received a Maser of Science in Mathematics from Osmania University, India, in 1993 and expects to receive a Master of Science in Computer and Information Sciences from the University of North Florida in December of 2006. Dr. Sanjay Ahuja of the University of North Florida is serving as Praveen's project director.

Praveen is currently employed as a senior software engineer at CSX Corporation and has been with the company for 8 years. Prior to that, Praveen worked for 3 years as a programmer analyst with Computer Management Services, Inc. in Jacksonville, Florida. Praveen has over 14 years of industry experience in Information Sciences.

Praveen’s interests are in real-time and parallel systems and distributed systems design and development. Praveen is married and lives with his wife and 6 year old son in Jacksonville.