FileAccessManager, a key management program including an implementation of RFC 2945, the SRP authentication and key exchange system

Katherine Clara Saunders Whitaker

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
Whitaker, Katherine Clara Saunders, "FileAccessManager, a key management program including an implementation of RFC 2945, the SRP authentication and key exchange system" (2002). Retrospective Theses and Dissertations. 21356.
https://lib.dr.iastate.edu/rtd/21356

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
FileAccessManager: a key management program including
an Implementation of RFC 2945: The SRP Authentication and Key Exchange System

by

Katherine Clara Saunders Whitaker

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Doug Jacobson (Major Professor)  
James A. Davis  
Clifford Bergman

Iowa State University
Ames, Iowa
2002
This is to certify that the master's thesis of

Katherine Clara Saunders Whitaker

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy
# Table of Contents

1. Introduction ............................................................................................................................................... 1
2. Background ............................................................................................................................................... 3
3. The SRP Protocol ...................................................................................................................................... 9
   3.1. SRP Protocol Details ........................................................................................................................ 9
   3.2. SRP Security and Performance Evaluation ...................................................................................... 10
4. FileAccessManager .................................................................................................................................. 12
   4.1. ManagerGUI ........................................................................................................................................ 13
   4.2. SHAInterleave .................................................................................................................................... 16
   4.3. SRPServer ......................................................................................................................................... 16
   4.4. SRPClient and ClientGUI ..................................................................................................................... 19
   4.5. Proof that Server Key and Client Key are equal ................................................................................ 22
   4.6. FileIDgui and UserListEditor ........................................................................................................... 22
   4.7. FAMcrypto ......................................................................................................................................... 25
5. Results ....................................................................................................................................................... 26
6. Future Work ............................................................................................................................................. 28
Appendix A. ManagerGUI.java ..................................................................................................................... 30
Appendix B. SHAInterleave.java .................................................................................................................... 39
Appendix C. SRPServer.java ......................................................................................................................... 41
Appendix D. SRPClient.java .......................................................................................................................... 53
Appendix E. ClientGUI.java .......................................................................................................................... 59
Appendix F. FileIDgui.java ............................................................................................................................ 62
Appendix G. UserListEditor.java ................................................................................................................... 72
Appendix H. FAMcrypto.java ........................................................................................................................ 75
References ..................................................................................................................................................... 78
1. Introduction

Small Company Example. Suppose Carl, Jane, and Rick work for the human resources department of a company and Sara is an employee of this company. The human resources department is working on plans to downsize the company. Sara would like to know if she is on the list of those who will be laid off. Carl, Jane and Rick need to be able to share electronic documents, but know that the company intranet is not secure. The human resources department decides that they need a way to share encrypted files. The department also knows that Rick uses simple passwords like his last name, because he tends to forget them if they are too complicated.

There are a number of solutions available to this human resource department. This paper proposes a solution that would require no additional hardware and would not require that Rick suddenly start using long and complicated passwords. For this work a program, FileAccessManager, was created. This program must be multiplatform for easy integration into any computer network, protect users with poor passwords like Rick, and not require an in depth knowledge of encryption from the users. Authentication to this program must be secure enough to keep an attacker from obtaining passwords for the users. The FileAccessManager creates a server that stores encryption keys with file identifiers and access lists. Authentication to the server is performed using the Secure Remote Password Protocol (SRP). Users of the FileAccessManager are not required to understand cryptography, they just need to know a username, password, which files they wish to protect and with whom they wish to share the files. The users are responsible for transmitting the documents to the other users, but this can be done easily with email or any other simple file transfer.

Before discussing the details of the FileAccessManager, an overview of authentication methods is necessary, with particular attention on the the evolution of SRP. Following an evaluation of SRP and
its security in Chapter 3 is an in depth look at the FileAccessManager package in Chapter 4. Chapter 5 contains the results of implementing the FileAccessManager and Chapter 6 contains a discussion of future work to expand the FileAccessManager.
2. Background

Now that computers are being used in business and in the home, there is a growing concern for users' privacy and security. Throughout the web, users are given a chance to enter a username and password allowing access to more personal information from a particular website. The use of Secure Sockets Layer (SSL) has improved security over the world wide web, but what about inside smaller networks. Not all local network administrators understand how to configure SSL, or maybe some applications are used in an environment unable to support SSL. A user's login name and password are sent across the network unencrypted. Any one who has access to the internal network could capture and read this information. Encrypting the information is one solution to preventing "casual" snoops from gathering useful information. Depending upon the strength of the encryption, even aggressive snoops could be denied access to the information. Over the years uses and methods of encryption and authentication have changed.

In their 1976 paper [3], Diffie and Hellman suggest some techniques for public key distribution systems. In this type of system, encryption keys are suggested and negotiated between two users, a user and a server or two different servers. A key is considered secure if it is "computational infeasible to compute the key from the information overheard"[3, p. 29]. This is the beginning of a new area of key exchange protocols. Bellovin and Merritt [2,1] developed the encrypted key exchange (EKE) and its extension, augmented encrypted key exchange (A-EKE). Following those exchanges, Jablon [6] offers the simple password exponential key exchange (SPEKE) and the Diffie-Hellman encrypted key exchange (DH-EKE) followed by an extension of SPEKE, B-SPEKE [5]. Finally Wu [13] offers SRP an asymmetric key exchange (AKE) as a secure authentication scheme. Before these protocols categories can be discussed, one of the most prevalent attacks against password authentication systems must be addressed.
Most key exchange protocols require that a user provide a password for authentication. One of the main security issues with this type of system is password selection. The user must be able to remember the password without writing it down. Even if the algorithm to compute the key is considered secure, if the password is weak the key may also be weak. One of the easiest attacks on an authentication system by a snooper is a "dictionary attack". This involves testing a list or dictionary of common words and passwords on the system either on- or off-line. Many systems log an error message and refuse to continue prompting for a username and password after a number of failed login attempts, usually 3. If the snooper can capture authentication messages and save them, he could then test each of the "passwords" in his dictionary off-line. By comparing decrypted versions of the saved messages with an expected type of plaintext, the snooper can discover the password. Because users generally choose passwords that are easily remembered or pronounceable words instead of random strings of letters, numbers and symbols, the likelihood that they appear in an snooper's dictionary is increased.

The first group of key exchange protocols to be discussed is EKE. EKE can protect against off-line dictionary attacks[2]. The protocol is a combination of asymmetric and symmetric cryptography. A new key distribution system is not required to implement EKE. Bellovin and Merritt [2] demonstrate the special case uses of both RSA and ElGamal in EKE in addition to a generic exponential key exchange version. Both the server and client need to know a shared password P. EKE works with the following generic exchange of messages between Alice and Bob. Alice first must generate a key E and then encrypt it using P. She then sends her name and this encrypted value, P(E), to Bob. He decrypts the key using his copy of P and generates a random number R. This value is then encrypted with both E and P to produce a message P(E(R)). When Alice receives this message, she decrypts it to get R. Then using R she encrypts a random challenge, challengeA. Bob receives this encrypted challenge, decrypts it, and generates his own challenge, challengeB. He then combines the two challenges, encrypts the result with R and sends R(challengeA, challengeB) to Alice. She verifies that the decrypted value for challengeA matches the value she created. If it does, she replies with
R(challengeB). Once Bob verifies the content of the message from Alice the authentication is complete[2]. This exchange can be diagramed as follows.

![Diagram](image)

**Figure 1. EKE**

Bob was required to have a plaintext version of P in order to verify Alice's identity. The value of P is stored at the server together with the client's user name. A system that stores a copy of the password or private key is called a *plaintext-equivalent* system.[13] A system that stores a password verifier instead is called a *verifier-based* system. Most implementations using EKE require the value of the raw password for initial encryption of messages. This means that brute force off-line dictionary attacks could be used to discover the password or the malicious party could get the password file from the server. Some versions of EKE provide forward secrecy, but they are only used with plaintext-equivalent systems.[12] Forward secrecy implies that if an snooper is able to calculate the current session key, this knowledge would not allow previous sessions to be revealed. While the snooper would not be able to decrypt previous sessions, he would be able to mount a dictionary attack on the password using the session key and then impersonate the legitimate user until the password was changed.[12] Using EKE with an exponential key exchange produces a practical variant that resists some of the attacks the generic EKE is vulnerable to.

Bellovin and Merritt [1] provide A-EKE as a verifier-based system that is protected against dictionary attacks. The changes that provide the protection against password file compromise make it
A-EKE substitutes a one way hash function of the password $H(P)$ for the password in EKE as illustrated below.

If an snooper were able to capture $H(P)$, he could impersonate the server, but could not impersonate the user because the value of the password is still unknown. By adding a salt value to the function, A-EKE is made more secure against dictionary attacks. Instead of using $H(P)$ as the verifier, $H(salt, P)$ is used. Both EKE and A-EKE will leak information to an snooper if either legitimate party is able to influence the selection of the session key.

Another key exchange protocol, Simple Password Exponential Key Exchange (SPEKE) offers the benefit of verifying that the user actually has the password, not a captured copy of the verifier. SPEKE and Diffie Hellman Encrypted Key Exchange (DH-EKE) offer protection against off-line dictionary attacks even when the passwords are very small. Neither system leaks any valuable information about the key during the authentication exchange. SPEKE breaks down the key exchange into two parts. First the parties establish a shared key using a Diffie Hellman exchange (DH). Once this key has been established, the two parties must verify that the other party has the same value for the key. This exchange is illustrated below.
Stage 1: Create session key

Client

\[ Q_a = S^{a} \mod p \]

Server

\[ Q_b = S^{b} \mod p \]

\[ K = h(Q_a \mod p) \]

\[ K = h(Q_b \mod p) \]

Stage 2: Verify session key

\[ \text{Random } C_a, E_k(C_a) \]

\[ \text{Random } C_b \]

\[ E_k(C_b) \]

\[ S = \text{secret} \]

\[ p = \text{large prime } h(x) = \text{one-way hash} \]

Figure 3. SPEKE

Both SPEKE and DH-EKE, when properly implemented with large modulus values, protect against discrete log attacks. SPEKE requires special consideration to ensure that the size of the discrete log problem is kept large.[6] Jablon[5] extends SPEKE by adding another DH exchange instead of a signature to prove that the user knows the correct password. This extension (B-SPEKE) produces a second key that should not be transmitted alone in a DH exchange. Such an action would open the exchange to a dictionary attack.[5] Instead both keys are transmitted together. DH-EKE provides a distinct advantage in producing a session key, both parties contribute equally to the creation of the key.[12]

An asymmetric key exchange (AKE) is another area of key exchange systems, but this area was designed for verifier-based systems. It also does not encrypt any of the messages used during authentication like previously discussed systems. Instead it uses mathematical relationships to verify the password. This verification method offers some protection in avoiding weaknesses of particular encryption algorithms. Also it allows for a wider spread implementation because implementations of this type of key exchange are not subject to import and export restrictions placed on various encryption algorithms.[13] Each party using AKE computes a secret, and from that a verifier. Just knowing the verifier is not enough to allow a snooper to impersonate the original user. The snooper must know the secret used to create the verifier. Four functions are used in this key exchange. One function is a one-way function used to create the verifiers from secrets while the other three functions are used to
combine the results of the first function with two publicly known parameters. This group of functions produce a session key for use during the data exchange for which the authentication was required.
3. The SRP Protocol

Wu [13] presents a new AKE authentication system. His Secure Remote Password Protocol (SRP) provides protection against weak or simple passwords and also protects the contents of previous and future sessions. A separate session key is computed for each authentication session as a strong encryption key. It cannot be guessed by a snooper, nor can it be calculated from publicly known values. The authentication system is not compromised even though the host server may have been completely compromised, because of the manner in which the password information is stored. [13]

Additionally with SRP there can be either mutual authentication (both the client and server authenticate to each other) or just the client can authenticate to the server, though this must be agreed upon ahead of time. If a malicious party is trying to gain access to a system using SRP he/she would be able to intercept messages, but can gather no useful information from the exchanges. Also he/she cannot impersonate a server because they lack the proper information to create responses to the client that will be accepted. Though with a stolen password verifier, he/she could impersonate a user until the password is changed.

3.1. SRP Protocol Details

The protocol starts with the server selecting a large prime number $N$ and a generator $g$. All computations are performed modulo $N$. Both $N$ and $g$ are required on both sides of the authentication. They can either be set in advance on both sides, or the server can send the values as part of the first response. The latter option allows the client to communicate with different SRP servers. For each client the server stores a username, random salt ($s$) and password verifier ($v$). The value of $v$ is computed using a hash of $s$ and the password.

$$x = H(s, \text{Password})$$

$$v = g^r \mod N$$
When the client begins the authentication process, it sends the user's name and the server looks up \( v \) and \( s \). The server sends \( s \) to the client. The client then computes its own \( x \) using the user's password. It then generates a random number, \( a \), computes \( A \) and sends that to the server.

\[
A = g^a \mod N
\]

The server generates its own random number, \( b \), computes \( B \) and sends that to the client. It also sends a randomly generated \( u \) to the client.

\[
B = v + g^b \mod N
\]

At this point both sides compute \( S \). This should produce the same result for both, assuming the client entered a correct password.

\[
S = g^{ab} \mod N
\]

A hash of \( S \) is performed to produce a session key. The client sends to the server its evidence that it has computed a session key. The server computes its own value of the session key and evidence and compares the resulting value with what the client sent. If the evidence matches, the server then sends its evidence to the client for verification. The resulting session key is then available for use for communication during the current session. This exchange as defined in RFC 2945 is illustrated in full below.

\[
\begin{align*}
\text{Client} & \quad \text{Server} \\
U & \quad s, N, g \\
a = \text{random} & \quad b = \text{random} \\
A = g^a \mod N & \quad B = (v + g^b) \mod N \\
x = \text{SHA}(s \text{SHA}(U) || P)) & \quad u = \text{first 32 bits of SHA}(B) \\
S = (B - g^y)^{x\mod 32} \mod N & \quad S = (A \cdot v^x)^{x\mod 32} \mod N \\
K = \text{SHA}(\text{Interleaves}(S)) & \\
M = \text{SHA}(\text{SHA}(N) \text{ xor SHA}(g) \text{SHA}(U)||s||A||B||K) & \\
M & \quad \text{SHA}(A||M||K)
\end{align*}
\]

\textbf{Figure 4. SRP}

3.2. SRP Security and Performance Evaluation

There are several security concerns in implementing SRP. Because SRP is a form of exponential key exchange like the Diffie-Hellman problem, its security relies on the difficulty of the discrete
logarithm problem. It should be difficult to compute $g^b \mod N$ given $g^a \mod N$ and $g^b \mod N$.\cite{13} Following this problem, $N$ should be at least 512 bits, though 1024 would be better. Allowing the implementation to change the size of $N$ increases flexibility as computation times for the discrete logarithm problem decrease. If the session key is disclosed to an attacker, combining it with the client's or server's evidence of the session key will not reveal non-public information. If the password verifier is leaked, the only thing the attacker can do is impersonate the server in an exchange, but not gather any password values.\cite{13} SRP is also immune to man-in-the-middle attacks, because the individual in the middle must know both the password verifier for one user and the password for the other.

Performance is always an important issue when considering altering programs for more security. Because of the mathematical calculations involved in verifier-based authentication systems, they are slower than plaintext-equivalent systems. Wu\cite{13} compares SRP to A-EKE, B-SPEKE, DH-EKE, and SPEKE. DH-EKE and SPEKE were used as indicators for plaintext-equivalent systems while the others are verifier-based. SRP is the fastest in its group. DH-EKE is the fastest overall. SRP also offers a computational advantage. If the modulo and generator are known ahead of time to both the server and client, $g^a$ and $g^b$ can be computed before authentication begins.
4. FileAccessManager

SRP authentication offers protection for simple passwords and also protection from replay attacks. This authentication system coupled with the FileAccessManager can offer encryption key management and file access protection. SRP is classified as a zero-knowledge protocol because it does not leak any information about the password to the host. The host knows that the password was correct, but does not know the plaintext version of the password. The host only sees the plaintext password at the time the user is added to the system.

The FileAccessManager uses SRP authentication for user verification to the server. Then using the key created with SRP, communicationbetween the user and server can be encrypted. The user can create a new File ID tag and encryption key with an access list or obtain the decryption key using the file ID tag. The FileAccessManager controls the keys and access lists for all file IDs in the system. If an unauthorized user attempts to access the system, he/she is denied at authentication before the program's main window appears. The FileAccessManager package consists of eight java programs: ManagerGUI, SHAInterleave, SRPServer, SRPClient, ClientGUI, FileIDgui, UserListEditor, and FAMcrypto. ManagerGUI is used to add users to the system. SRPServer and SRPClient negotiate authentication using SHAInterleave. ClientGUI provides a hidden password entry box for starting authentication. FileIDgui is the main program for managing file keys and access lists and uses the UserListEditor.

Before the FileAccessManager can be used, the users must be added to the system. All of the users of the network can be added or just those users who will be using the FileAccessManager can be added. The ManagerGUI program provides that service. The system administrator must first run this program to ensure that all of the proper users are added with their correct passwords. This information is stored in a separate file, 'userList.txt'. Having access to this file will not expose the users' passwords, because they are stored as {username, password-verifier, salt} triplets. It is recommended that a special
FileAccessManager user be created on the computer system on which the Manager and Server will run. This user should be the only one able to access the 'userList.txt' and 'access.txt' files and run the Manager and Server.

Once a user is in the FileAccessManager system, he/she does not need to understand how the actual encryption works. Because the user need not understand encryption, the FileAccessManager can be used in any computing environment. To use the system, the user only needs to know his/her username, password and the files that need to be either encrypted or decrypted. The FileAccessManager can be used in any sort of networking environment from a home user who wishes to encrypt files on his/her personal computer to a large corporation that needs extra protection for a special set of documents or department. Because this system has not undergone strict security tests, it is not recommended that it be used as the primary protection for files. As always it is the user's responsibility to ensure that sensitive material not be left available to unauthorized individuals.

4.1.ManagerGUI

This Java program allows for administration of the user and password verifier file. It is a graphical program allowing the administrator to add users to the system, view a list of users or view the entire file. The main window contains a username text area, password field, add user button, list users button, show file button, message text area and exit button.

All file access is completed using a java.io.RandomAccessFile object. Using this object allows for reading and writing to a file without deleting the existing contents of the file.
To add a user to the file, a username and password are required. The password is entered in a password field with '*' echoed to the screen for each character in the password. Both username and password are case sensitive. The Manager first checks the existence of the file 'userList.txt'. This file must be in the current working directory. If that file does not exist, it is created along with new values for the safe prime \( (N = 2q + 1) \) and a primitive root modulo \( N \) \((G)\). The variable \( q \) is created using \( \text{Java.math.BigInteger.probablePrime} \) with 512 bits and a random number. The probability that \( q \) is a composite number "does not exceed \( 2^{-100} \)"[7] \( \). N is then computed from \( q \), where \( N = 2q + 1 \). G is then created as a random number that is tested as a generator mod \( N \). G is a generator if:

\[
G^q \equiv 1 \mod N \quad \text{and} \quad G^{\varphi(N)} \equiv 1 \mod N
\]

[10, p. 254]

These two values are then written to 'userList.txt' with the following pattern, spaces separating each piece and a space at the end:

\[
N \text{ <value>} G \text{ <value>}
\]

Next a secure random number is generated by a Secure Hash Algorithm (SHA) pseudo random number generator (PRNG) as the salt, \( s \). Then a string with the value of the user name, "::" and password is created. An SHA hash is then computed over this string and appended to the end of the string.
containing the salt. Another SHA hash is then computed over the resulting string. This 160 bit value is then checked for inversion and if it is not invertible, a new salt is selected and the hash process repeated until it is invertible.

\[ x = SHA(s | SHA(username : : password)) \]

A password verifier, v, is computed using the x, N and G.

\[ v = G^x \mod N \]

These values are then written as strings to the file userList.txt surrounded by braces and separated by a comma and a space, and ending with a space.

\{username, v, s\}

If an arithmetic exception is thrown, the program chooses another value for s. If any other error occurs, an exception is thrown and the program prints an error message to the screen.

A list of the current users in 'userList.txt' is available by pressing the corresponding button. This method reads 'userList.txt' into a StringBuffer and then searches for the first occurrence of "<space>{". This indicates the first user in the file. The string between the first { and the next space is copied to a new StringBuffer and a new line character is appended to the end of it. The starting location for the next "<space>{" search is set to the first character in the username and the search is repeated until the end of the file. The String containing the usernames is then printed in the message text area of the main window.

The Manager also provides the administrator with the option to view the entire contents of 'userList.txt' formatted nicely. The file is printed with the values for N and G and the first user on the first line of the message text area and then each subsequent user is listed on a separate line. This can be used for either debugging the system or for verifying a user's information.
4.2 SHAInterleave

The SHAInterleave method is in a separate class because it is used both by the Server and Client in creating the session key. This method takes a java.Math.BigInteger (Input) as a parameter and returns a 320 bit java.math.BigInteger. After converting Input to a byte array with the most significant bit in byte[0], any leading 0 bytes are removed. Then the number of bytes in the array are counted. If that is an odd number the first byte of the array is also removed. Alternating destinations, the array is split byte by byte into two smaller arrays, E and F. E contains the even numbered bytes and F the odd numbered bytes. SHA is computed on E and F producing 20 byte arrays G and H. A 40 byte output array is constructed by alternating bytes from G and H with the first byte[0] = G[0]. The output array is then converted back into a java.math.BigInteger and returned to the calling method.

4.3 SRPServer

The Server is run from the command line with no parameters. The directory from which it is run must contain 'userList.txt' and 'access.txt', if the second file exists. The Server first reads the values of N and G from 'userList.txt' and then waits for the Client to connect to port 50505. This port was chosen because ports 0 to 1023 are used by well-known services, ports 1024 to 49151 are for services registered with Internet Assigned Numbers Authority (IANA) and ports 49152 to 65535 are dynamic or private ports.[9] If 'userList.txt' and/or N and G do not exist (or were unable to be read correctly) the Server prints out an error message. Otherwise the Server reads the values for N and G and stores them in global variables. It then listens for connections to the socket. When a Client connects, the Server starts a new instance of the SRPServer class in a separate thread allowing the Server to have multiple connections. The incoming connection from the Client must provide the username of the client. The Server thread then looks up the client in 'userList.txt'. This lookup is case specific. The values corresponding to the username are stored in the global variables String U, BigInteger passVerifier and BigInteger salt. The Server then sends the salt to the Client. If the user is not in 'userList.txt', the
Server sends "Invalid username" and records the invalid login in the command window. After sending salt, the values for N and G are also sent to the Client. The Server then waits until the Client sends A. After checking that the response from the Client was not "abort", the value received is stored in A. If \( A = 0 \) or \( A \% N = 0 \), the Server responds with "abort" signaling that an error occurred and closes the thread. Otherwise, the Server selects a random value \( b \) that is 64 bytes long using an SHA PRNG. The variable \( B \) is then calculated.

\[
B = (v + G^b) \% N
\]

If this value is calculated without error, it is sent to the Client, otherwise "abort" is sent and the thread closed. The Server then calculates a value for the variable \( u \) from the first 32 bits (most significant bit first) of the SHA hash of \( B \). \( S \) is then calculated from \( B, u, A, v \) and \( N \).

\[
S = (A \times v^u)^b \% N
\]

The SHAInterleave function is then performed on \( S \) to produce a 40 byte session key, \( K \). If there is an error in calculating \( K \), the Server sends "kerror" to the Client and then waits for A to be resent and repeats the authentication process from that point. This restart includes creating another random value for \( b \) and recalculating \( B \). Likewise if the Client sends "kerror" the Server resets the authentication and waits for a new value for \( A \) to be sent. Any other errors encountered along the way cause the Server to send "abort" to the Client and close out the thread. The Server now listens for a response from the Client. If the response is "abort" the thread is closed. Otherwise the response is stored in ClientM.

The Server then calculates ServerM by first producing hash values from \( N \) and \( G \). These values are then exclusive or-ed. The result is concatenated into a string with the hash of the username, the string representation of the salt, \( A, B \) and \( K \).

\[
serverM = H(H(N) \text{xor} H(G)|H(\text{username})|\text{salt}|A|B|K)
\]

If this value equals clientM the Server responds with a simple hash of \( A \), serverM and \( K \) concatenated.

\[
Response = H(A|M|K)
\]
This final response is required for the Client to verify K to authenticate the Server. If serverM did not equal clientM, the Server would respond with "abort" signaling an error in the user's password or in the calculations on the Client side. The Server would also print the invalid login attempt to the command window. Once the Client verifies K, it can be used as a session key for encrypting communications. The Server then prints the username and authentication complete to the command window.

Now that authentication is complete, the Server then opens the file 'access.txt' and reads in the last file identifier listed. If the file does not exist, it is created and the variable nextFileID is set to 1. The file 'access.txt' contains the filename, identifier, encryption key, owner and access list for each file in the system. This information is stored as strings. Each entry is formatted as follows:

{fileID, filename, key, owner, string of users separated by spaces}

The file identifiers are ordered sequentially. Every time a new file is added to 'access.txt', nextFileID is incremented. This numbering scheme allows for easy checking when verifying that information is stored for a particular fileID.

Next, the Server listens for requests from the Client regarding file identifiers and encryption keys. If the Client sends "Get Key", the Server then stores the next value received as the filename for which an identifier and key must be produced. The next value sent by the Client is the number of users in the access list. Because each user is sent separately, the Server needs to execute a loop reading in usernames and storing them in a StringBuffer separated by spaces.

The key is generated as a data encryption standard (DES) key. This was chosen because the algorithm was supplied in the Sun java cryptographic extension (SunJCE) that is included in the java platform. A new instance of a key generator must first be created and then a secret key can be generated. That secret key is then used to create a DES key specification that is required by the secret key factory. The factory then produces a DES key as a byte array. This is then converted to a BigInteger. Once the key has been created, the file information must be written to 'access.txt'. The
fileID and key are also sent to the Client. If any errors occur during the creation or writing of the key, "error" is sent to the Client and an error message is printed to the screen.

If the incoming request is "Get File Info", the Server reads the next value as the fileID for which the information is requested. If that fileID is equal to or larger than nextFileID, it immediately returns "FileID not found". It also checks to see if 'access.txt' is smaller than 10 bytes, indicating that the file is too small to contain any file information. "FileID not found" is returned in this case as well. If 'access.txt' is larger than 10 bytes and the fileID is less than nextFileID, characters are read in from 'access.txt' until the first instance of '{' occurs. At that point, the first digit following '{' is compared to the first digit of the fileID, if those match the rest of the fileID value is read from 'access.txt' and compared with the requested fileID. If the fileID values match, the filename is then read as the next string of characters that ends with '}'. This read process is repeated to retrieve the key and owner. The last remaining information about the file is the string of usernames separated by spaces. The Server reads this information in until it reads '}' indicating the end of information for that particular fileID. The client's username is then compared with the string of usernames to verify that the user is allowed access to the file information. If the client's username appears in the list, the information is then sent. Otherwise, "access denied" is sent to the Client. If the first digit of the fileID did not match the initially read character, the Server reads until the next occurrence of '{' and repeats the above procedure. The fileID should be found in 'access.txt' unless an error occurred.

Any errors that occur during the HandleFileRequests method cause the Server to send an "Error" message to the Client, the method to return and the thread to be terminated. Also receiving "exit" from the Client will cause the thread to close the connection and then terminate.

4.4.SRPClient and ClientGUI

ClientGUI is the main class in the fileAccessManager package. This class contains a main method that initiates the authentication. The server name or IP address is required as a parameter. The
program begins with a frame that contains a username text field, a password field, message text field, an ok button and a cancel button.

![Figure 6. ClientGUI](image)

The password field echoes '*' for each character entered into it. If the cancel button is pressed, the program exits without contacting the Server. When the ok button is pressed, the username and password are gathered from the text area and password field and passed along with the server name to the SRPClient.Authenticate method.

SRPClient.Authenticate first attempts to connect to the Server as given through port 50505. Then a PrintWriter, InputStreamReader and BufferedReader are created on that socket. These allow for reading and writing information to the Server. The Client first sends the username to the Server to initiate the authentication process. It then waits until the Server sends the salt stored with the username in 'userList.txt'. If the Client receives "Invalid" from the Server, it prints an error message in an information dialog box, closes the socket and exits the authentication. If the received value from the Server was not "Invalid", the value is stored as salt and the next two values from the Server are stored as N and G respectively.

The Client next calculates the variable A from a random number a, N and G.

\[ A = G^a \% N \]
This value is then sent to the Server. The Client next computes a value for \( x \). This is the same computation that the Manager did while creating the password verifier. It first produces a string by concatenating the username, ",:", and raw password. The SHA hash of the string is then computed and appended to a string containing the salt. A final SHA hash is performed on the entire string to produce a 160 bit value for \( x \).

\[
x = SHA(s | SHA(username : password))
\]

The Client waits until the Server sends a value for the variable \( B \). It then checks if \( B \% N = 0 \). This would result in aborting the authentication by sending "abort" to the Server and then closing the socket and exiting the authentication process. The Client now has all the information required to compute \( S \) and from that the value for the session key, \( K \). \( S \) is computed using \( B \), \( G \), \( x \), \( a \), \( u \) and \( N \). First the value of \( u \) must be computed from the first 32 bits of the SHA hash of \( B \), most significant bit first.

\[
S = (B - G^x)^u \% N
\]

This value is computed using different variables from the Server. The resulting value for \( S \) is then passed the the SHAInterleave method and the end result is a 40 byte session key, \( K \). If an error occurs during the calculation of \( S \) or \( K \), the Client sends "kerror" to the Server and resets the authentication by generating a new value for \( a \) and recalculating \( A \) to send to the Server. If the Server encountered an error while calculating \( S \) or \( K \), it sends "kerror" to the Client indicating that a reset of the authentication process is necessary. In order to prove to the Server that the user has correctly authenticated, the Client must compute a 20 byte value \( M \) by producing the SHA hash of a the following string:

\[
M = H(H(N) xor H(G)) | H(username) | salt | A | B | K
\]

The Client sends this value to the Server and then waits for proof from the Server that \( K \) is the session key. Once this proof has been received the Authenticate method returns to ClientGUI. From there a new instance of FileIDgui is called.
4.5. Proof that Server Key and Client Key are equal

The Server and the Client produce values for the variable S using different formulas. The password is verified when these values are equal. The Server uses the following equation for S.

\[ S_s = (A \cdot v^a)^b \mod N \quad \text{where} \quad A = g^a \mod N \]

Then substituting the value for A, the following equation is produced.

\[ S_s = ((G^a \mod N) \cdot v^a)^b \mod N \quad \text{where} \quad v = G^x \mod N \]

Then after substituting the equation for v, S now is stated with the following equation.

\[ S_s = ((G^a \mod N) \cdot (G^x \mod N)^a)^b \mod N \]

The terms are combined and the following equation results.

\[ S_s = G^{ln + bx} \mod N \]

The Client uses the following equations to produce its version of S.

\[ S_c = (B - G^z)^{x + x} \mod N \quad \text{where} \quad B = (v + G^b) \mod N \]

After substituting the values for B and v, the Client has the following equation.

\[ S_c = ((v + G^b) \mod N - G^x \mod N)^{x + x} \mod N \]

Terms are combined and the Client produces the equation:

\[ S_c = (G^b)^{x + x} \mod N \]

After combining the exponents the Client produces the same equation as the Server, thus proving that both the Server and Client will compute matching values for S.

\[ S_c = G^{ln + bx} \mod N = S_s \]

4.6. FileIDgui and UserListEditor

FileIDgui provides the main window for the FileAccessManager. It is called after the authentication process. It first disposes of the ClientGUI that called it to free up resources and then initializes its components. The main window contains text fields in which the filename, fileID or key can be entered, a text field for messages from the Server, a text area containing the access list for the
currently selected file and buttons to retrieve file information, set a key, edit the access list, clear the form, encrypt a file and decrypt a file.

![FileID Manager Main Window](image1)

**Figure 7.** FileIDgui main window

The text area initially contains the username as the only entry in the access list. When the edit access list button is pressed a new window appears containing the current list of users from the text area in the main window.

![UserListEditor Window](image2)

**Figure 8.** UserListEditor window

The user is given the option to enter names into a new text area, clear the current access list, cancel the edit operation or save the edited access list. When new users are added to the access list they should be
on separate lines. The access list can be edited at any time prior to pressing the Get Key button when
creating a new file entry into the system.

In order to obtain a key and fileID for a file, the user must edit the access list, enter a filename
and then press the Get Key button. The file referenced by the filename must be in the current working
directory. The Get Key button sends first the command "Get Key" then the filename followed by the
number of users in the access list. Each of the users is then sent separately. Once the information is
sent, fileIDgui waits for the Server to respond with the key and fileID. The fileID is printed in the
corresponding textfield in the window and "key obtained" is printed in the Key textfield. Once the user
has set the key for a file, the encrypt file button can be pressed. FileIDgui first opens a confirmation
dialog asking the user to verify the filename and encryption process and then sends the filename,
fileID, key and access list to the encrypt method in the FAMcrypto class. This encryption process
creates two files in the current working directory; fileIDenc and fileIDIV. When the user wishes to
share the encrypted file with other users, these two files must be sent together.

When the user wishes to decrypt a file, the user must first place the files fileIDenc and fileIDIV
into the current working directory. The fileID must be entered in the appropriate text field and then the
"Get file info" button should be pressed. This sends the fileID to the Server. The Server must check to
see if the fileID exists and also that the user is on the access list for the given file. If the user is
authorized to access the file, the Server returns the original filename, key and access list. This
information is written to the main window in the appropriate locations with "key obtained" printed in
the key textfield and the message "File information retrieved" is printed in the message text field. The
user should then press the "Decrypt file" button. This action opens a confirmation dialog before calling
the FAMcrypto decryption method.

If the clear form button is pressed, the text fields are set to blank text and the access list text area
is reset to contain only the current user. When the exit button is pressed or exit is chosen from the
program menu, Client sends an "exit" message to the Server signaling that the socket and thread can be closed. The Client then closes its socket and exits.

4.7. FAMcrypto

The FAMcrypto class contains two methods, one for encrypting files and the other for decrypting files. Both methods use DES in cipher feedback mode with a block size of 8 bits. Because the encryption and decryption are performed on cipher streams, an 8 bit (1 byte) block size ensures that the stream is not held while a full block is collected before performing the encryption. This form of DES is included in the SunJCE. Both methods use two files, whose names are constructed from the fileID value. The first file is the fileID followed by "enc". The other file is the fileID followed by "IV". For example, if the fileID was 15, the two files would be '15enc' and '15IV'.

The encryption method initializes the cipher in encryption mode with the key provided by the calling method. It creates two files. The fileIDIV contains the original filename of the file that is being encrypted and the initialization vector that will be used in decryption. The other file, fileIDenc, will contain the encrypted data. The input file is read in as a stream and passed to the cipher output stream that performs the encryption and writes the data to fileIDenc. Any errors that occur are sorted by type. All errors print appropriate messages to the screen and return an error value to the calling method.

The decryption method first looks for two files in the working directory. The enc file contains the actual encrypted file, while the IV file contains the original filename and the initialization vector for the DES decryption. A new output file named the same as the original filename of the encrypted file is created. The cipher object is initialized with decryption mode, the key and the initialization vector. Then using a cipher output stream and a file input stream the encrypted file is read, decrypted and written to the output file. If errors occur during the decryption process, messages will appear in the calling window and this method will return an error value to the calling method.
5. Results

The FileAccessManager was tested with the server running on RedHat 7.2 and the Java 2 Platform version 1.4.0. This platform includes the SunJCE. The client was also run on RedHat7.2 and on Windows 2000 Professional with the Java 2 Platform version 1.4.0. All of the files required for the FileAccessManager are contained in the package, fileAccessManager. To run the Manager for the first time the following command is used from a command prompt.

$ java fileAccessManager.ManagerGUI

A window will open that contains all of the information necessary to enter users into the system. Error messages will print to both the window and the command window, while debug information will print to just the command window.

The Manager must be run before SRPServer is run and both must be run from the same working directory to ensure that the file 'userList.txt' is available. The server is run with the command:

$ java fileAccessManager.SRPServer

The server runs solely in a command window. Debug information is printed to the window during both authentication and handling file requests. The working directory must also contain the file 'access.txt', though this will not exist the very first time the server is run. This file should never be deleted unless the administrator wishes to reset the entire system because encryption keys will be permanently lost. It is recommended that the administrator create backup copies of both 'userList.txt' and 'access.txt'.

The client is also run from a command window.

$ java fileAccessManager.ClientGUI <server name>

The server name can either be a host name or a host internet protocol (IP) address. The client prints debug information to a message text field in the main window. Error messages are also printed to the message text area of the fileIDgui after authentication. Files that are to be encrypted or decrypted must be located in the working directory and their names cannot contain spaces.
Because performance is an issue in computing today, tests have been run on both operating systems. Windows 2000 professional was running on an Intel Pentium 3 450 Mhz processor and RedHat 7.2 was running on an Intel Pentium 2 350 Mhz processor. The amount of time required for the manager to start, including creation of the values for N and G was noted, along with the amount of time required to add a user to the system. Then the time required to start SRPServer and time required to process authentication on the server was recorded. This process was tested first with all programs running on RedHat 7.2, then with all programs running on Windows 2000, and then with SRPServer and ManagerGUI on RedHat 7.2 and the client on Windows 2000. The time was recorded by printing to the screen the system time. Printing to the screen did increase the time of execution. Each run was performed 5 times and the values averaged.

<table>
<thead>
<tr>
<th>Platform (client → server)</th>
<th>Start</th>
<th>Add first user</th>
<th>Add second user</th>
<th>Start Server</th>
<th>Authenticate</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedHat → RedHat</td>
<td>18778.25 ms</td>
<td>529.75 ms</td>
<td>314.5 ms</td>
<td>162.0 ms</td>
<td>13948.25 ms</td>
</tr>
<tr>
<td>W2K → W2K</td>
<td>32298.0 ms</td>
<td>5733.25 ms</td>
<td>80.0 ms</td>
<td>75.25 ms</td>
<td>11696.75 ms</td>
</tr>
<tr>
<td>W2K → RedHat</td>
<td>26226.25 ms</td>
<td>400.0 ms</td>
<td>80.5 ms</td>
<td>173.75 ms</td>
<td>8883.0 ms</td>
</tr>
</tbody>
</table>

*Table 1: Timing Results*

The FileAccessManager provides security for users passwords by using SRP as the authentication protocol. The security of the entire system depends on the discrete logarithm problem. With the value for the large prime N set to be 1024 bits in length, the discrete logarithm problem is computationally infeasible with current computing power. While simple passwords are protected by SRP, it is recommended that passwords contain a mix of characters, numbers and symbols to further protect the system.
6. Future Work

In the future, FileAccessManager can be modified to handle multiple Clients at one time. The primary change to be implemented for this modification is file blocking to prevent two different threads of the server from reading or writing to 'userList.txt' or 'access.txt' at the same time.

ManagerGUI can be modified to allow users to be deleted from the system and to allow users to change their passwords. Currently only the administrator who would run ManagerGUI is able to enter a user to the system, setting the password for the user, though this data entry could be done in the presence of the user allowing he/she to type a personal password. Because a password field is used, the administrator would not be able to see the password value on the screen. In the future it would be advantageous to allow the authenticated user to change his/her password from the fileIDgui program.

Because a short performance time is preferred, a few modifications would be necessary. The first would be possibly changing the language in which the program is written on the server side. The server program could be optimized for the hardware used. The client still needs to be platform independent for ease in use, so it should not be translated.

Creating different levels of security given the lifetime of the data would also improve the speed of the program. When creating a file ID, the user would also include a date at which the data would expire and a secrecy level. These levels would be set by the administrator to different encryption algorithms and key lengths to protect the data. If the system frequently handles temporary fileIDs, faster and less strong algorithms could be used for data encryption.

Wu [13] offers a version of condensed SRP that reduces the number of messages sent between the client and server during authentication. The Client sends both a username and the value for A. The Server responds with the salt and the value for B. The Client computes M and sends that. The Server verifies M and responds with agreeing with the session key. SRP can also be used in a one way
authentication process, where the users trust the server. In this case the server does not need to respond agreeing with the session key.

Some other considerations for future work on the FileAccessManager include authentication to run the Manager. The Manager would have a preassigned administrator username and password. The administrator would be required to change the password the first time the Manager is run. Also the locations of 'userList.txt' and 'access.txt' would be changed to be located in an absolute path instead of a relative path to the working directory. This change would also allow files being encrypted and decrypted to be located in different directories. A pull down directory would be used to locate files in fileIDgui. For additional security, the sessions should have a time limit before requiring reauthentication. This will prevent a user from starting the program and then walking away from a session without shutting down the connection allowing a malicious user to use the program. For more security on the Server, error messages would be output to an error log file and user access logged to another log file.
/*
 * ManagerGUI.java
 */

package fileAccessManager;
import java.io.*;
import java.net.*;
import java.security.*;
import javax.net.*;
import java.math.*;
import java.util.*;

public class ManagerGUI extends javax.swing.JFrame {

    public static BigInteger N;
    public static BigInteger G;
    private static int nLength = 128;
    private static int gLength = 128;
    private static int saltLength = 32;
    public static RandomAccessFile userFile;
    private static boolean noN;
    public static Calendar myTime;

    /** Creates new form ManagerGUI */
    public ManagerGUI() {
        initComponents();
    }

    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        bgPanel = new javax.swing.JPanel();
        unLabel = new javax.swing.JLabel();
        unText = new javax.swing.JTextField();
        passLabel = new javax.swing.JLabel();
        passField = new javax.swing.JPasswordField();
        addUserButton = new javax.swing.JButton();
        listButton = new javax.swing.JButton();
        showFile = new javax.swing.JButton();
        messageLabel = new javax.swing.JLabel();
        messageSPane = new javax.swing.JScrollPane();
        messageText = new javax.swing.JTextArea();
        exitButton = new javax.swing.JButton();

        setTitle("User Manager for the FileAccessManager");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

    }

}
bgPanel.setLayout(new java.awt.GridBagLayout);
bgPanel.setMinimumSize(new java.awt.Dimension(1010, 500));
bgPanel.setPreferredSize(new java.awt.Dimension(1010, 500));
unLabel.setText("Username");
gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bgPanel.add(unLabel, gridBagConstraints);
unText.setMinimumSize(new java.awt.Dimension(69, 20));
unText.setPreferredSize(new java.awt.Dimension(69, 20));
gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 0;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bgPanel.add(unText, gridBagConstraints);
passLabel.setText("Password");
gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bgPanel.add(passLabel, gridBagConstraints);
passField.setMinimumSize(new java.awt.Dimension(69, 20));
passField.setPreferredSize(new java.awt.Dimension(69, 20));
gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bPanel.add(passField, gridBagConstraints);
addUserButton.setText("Add user");
addUserButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        addUserButtonActionPerformed(evt);
    }
});
gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bPanel.add(addUserButton, gridBagConstraints);
listButton.setText("List Users");
listButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        listButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bgPanel.add(listButton, gridBagConstraints);

showFile.setText("Show userList file");
showFile.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        showFileActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bgPanel.add(showFile, gridBagConstraints);

messageLabel.setText("Messages");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bgPanel.add(messageLabel, gridBagConstraints);

messageSPane.setMinimumSize(new java.awt.Dimension(800, 300));
messageSPane.setPreferredSize(new java.awt.Dimension(32767, 300));
messageText.setText("Welcome to the user manager for FileAccessManager");
messageText.setMaximumSize(new java.awt.Dimension(32767, 32767));
messageText.setMinimumSize(new java.awt.Dimension(800, 300));
messageSPane.setViewportView(messageText);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 3;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.gridheight = 5;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
bgPanel.add(messageSPane, gridBagConstraints);

exitButton.setText("Exit");
exitButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        exitButtonActionPerformed(evt);
    }
});
exitButtonActionPerformed(evt);
}
});//end initComponents

private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}

private void showFileActionPerformed(java.awt.event.ActionEvent evt) {
    StringBuffer myBuf = new StringBuffer();
    messageText.setText(""");
    int start = 0;
    int stop = 0;
    try {
        userFile.seek(0);
        while (userFile.getFilePointer() < userFile.length()) {
            myBuf.append(userFile.readChar());
        }
        String filetext = new String(myBuf.toString());
        while (start < filetext.length()) {
            stop = filetext.indexOf(" {", start);
            if (stop < 0) {
                messageText.append(filetext.substring(start) + 
                    
            start = filetext.length();
            } else {
                String mySubString = new String(filetext.substring(start, stop));
                if (mySubString.startsWith("N")) {
                    int space = mySubString.indexOf(" G", 0);
                    messageText.append(mySubString.substring(0, space) + "
            messageText.append(mySubString.substring(space + 1) + "
                
            start = stop + 1;
        }
    } //end while not at end of file
    } catch (IOException ioe) {
        messageText.setText("Error reading userFile " + ioe.getMessage());
    }
}
private void listButtonActionPerformed(java.awt.event.ActionEvent evt) {
    StringBuffer myBuf = new StringBuffer();
    messageText.setText("Users:\n");
    int start = 0;
    int stop = 0;
    int end = 0;
    try {
        userFile.seek(0);
        while (userFile.getFilePointer() < userFile.length()) {
            myBuf.append(userFile.readChar());
        }
        String filetext = new String(myBuf.toString());
        start = filetext.indexOf(" ");
        if (start > 0) {
            start += 1;
            while (start < filetext.length()) {
                stop = filetext.indexOf(" ", start);
                end = filetext.indexOf(" ", start);
                if (stop < 0) {
                    messageText.append("\t"
                        + filetext.substring(start + 1, end) + '\n');
                    start = filetext.length();
                } else {
                    messageText.append("\t"
                        + filetext.substring(start + 1, end) + '\n');
                    start = stop + 1;
                }
            }
        } catch (IOException ioe) {
            messageText.setText("Error reading userFile " + ioe.getMessage());
        }
    }
}

private void addUserButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (addUser(unText.getText(), new String(passField.getPassword())) == -1) {
        messageText.setText("Error adding user");
    } else {
        messageText.setText("User " + unText.getText() + " added");
        unText.setText(""),
        passField.setText("");    
    }
}

private void exitForm(java.awt.event.WindowEvent evt) {
    System.exit(0);
}

public static void main(String args[]) {
    // Try to open userList.txt file. Then read the first char to see if
    // N has already been set for the system. If it has then read in the
// values for both N and G.
myTime = Calendar.getInstance();
System.out.println(myTime.getTimeInMillis());
int spot = 0;
int succeeded = 0;
noN = true;
// open the file If it is not found then exit the program
try {
    userFile = new RandomAccessFile("userList.txt", "rw");
} catch (FileNotFoundException fnfe) {
    System.exit(0);
}
// read N and G from userList.txt
getNandG();
//if N and G don't already exist in userList.txt create them now
if (noN == true) {
    createNandG();
    getNandG();
} //end noN == true
// now open the GUI
new ManagerGUI().show();

/**
 * getNandG() looks to see if the file "userlist.txt" already exists and if
 * it contains readable values for N and G by looking for the pattern:
 * N <value> G <value> _ = the string " 
 */
public static void getNandG() {
    int spot = 0;
    int i = 0;
    char myChar;
    try {
        //File has "N value G value" followed by "{username passverifier
        //salt}" for each user separated by a space
        userFile.seek(spot);
        myChar = userFile.readChar();
        if (myChar == 'N') {
            spot += 4;
            userFile.seek(spot);
            StringBuffer nsb = new StringBuffer();
            myChar = userFile.readChar();
            while (myChar != ' ') {
                nsb.append(myChar);
                myChar = userFile.readChar();
            }
            N = new BigInteger(nsb.toString());
        }
        myChar = userFile.readChar();
        if (myChar == 'G') {
            myChar = userFile.readChar();
            StringBuffer gsb = new StringBuffer();
            myChar = userFile.readChar();
            while (myChar != ' ') {
                gsb.append(myChar);
                myChar = userFile.readChar();
            }
            G = new BigInteger(gsb.toString());
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
*/
gsb.append(myChar);
    myChar = userFile.readChar();
}
    G = new BigInteger(gsb.toString());
    noN = false;
}
catch (Exception raf) {
}
}//end getNandG()

/** createNandG() creates the values for the large prime N and a generator G
* these values are then written to the file "userlist.txt" following the
* the pattern:
*    N<value>_G<value>_ where _ = the string "
*/
public static void createNandG() {
//create N and g Store these in the beginning of the user file
    int bits = nLength * 8 - 1;
    Random myRandom = new Random();
    BigInteger q = BigInteger.probablePrime(bits, myRandom);
    N = q.multiply(BigInteger.ONE.add(BigInteger.ONE));
    N = N.add(BigInteger.ONE);
    try {
        userFile.seek(0);
        userFile.writeChar('N');
        userFile.writeChar(' ');
        userFile.writeChars(N.toString());
    } catch (IOException ioe) {
}
    // create generator G
    // pick random number < N-1 and test it if it is a generator then
    // write it to userList.txt
    byte temp[] = new byte[gLength];
    do {
        myRandom.nextBytes(temp);
        G = new BigInteger(temp);
        if (G.signum() == -1) {
            myRandom.nextBytes(temp);
            G = new BigInteger(temp);
        }
        while (G.toString().charAt(0) == '1') {
            myRandom.nextBytes(temp);
            G = new BigInteger(temp);
        }
    } while ((isGenerator(G, q) == false) && (G.compareTo(N) < 0));
    try {
        userFile.writeChar(' ');
        userFile.writeChar('G');
        userFile.writeChar(' ');
        userFile.writeChars(G.toString());
        userFile.writeChar(' ');
    } catch (IOException ioeG) {
}
** isGenerator returns true if g is a generator and false if not
* tests g to see if it is a generator for N where N = 2q + 1
* factors of N - 1 are then 2 and q
* g is a generator if g^q %N != 1 and g^2 % N != 1
**/

```java
public static boolean isGenerator(Biglnteger G, Biglnteger q) {
    boolean generator = false;
    Biglnteger g = G;
    Biglnteger two = new Biglnteger("2");
    if (Biglnteger.ONE.compareTo(g.modPow(q, N)) != 0) {
        if (Biglnteger.ONE.compareTo(g.modPow(two, N)) != 0) {
            generator = true;
        }
    }
    return(generator );
}
```

** addUser takes a userName and raw password, creates a random salt
* and computes a password verifier using the raw password and salt
* the username, passVerifer and salt are written to the file "userlist.txt"
* in the form:
*   {<username>, <passVerifier>, <salt>}
* with a space at the end.
**/

```java
public static int addUser(String userName, String passwd) {
    int succeeded = 1;
    Biglnteger passVerifier;
    boolean retry = false;
    byte tempSalt[] = new byte[saltLength];
    byte saltB[] = new byte[saltLength+1];
    Biglnteger salt;
    do {
        try { 
            SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
            random.nextBytes(tempSalt);
            saltB[0] = (byte)0;
            for (int i = 0; i < saltLength; i++)
                saltB[i+1] = tempSalt[i];
            salt = new Biglnteger(saltB);
            MessageDigest md = MessageDigest.getInstance("SHA");
            String userPass = new String(userName);
            userPass = userPass.concat(":");
            userPass = userPass.concat(passwd);
            md.update(userPass.getBytes());
            byte temp[] = md.digest(); //SHA (userName + ":" + passwd)
            md.reset();
            String saltStr = new String(salt.toString());
            saltStr = saltStr.concat(new String(temp));
            md.update(saltStr.getBytes());
```
byte smallX[] = new byte[20];
smallX = md.digest();
BigInteger x = new BigInteger(smallX);

// check to see if x is invertible if not then an arithmetic exception is thrown catching it retries with a new salt
BigInteger y = x.negate();
passVerifier = G.modPow(x, N);
// write the user information to the end of userList.txt
userFile.seek(userFile.length());
userFile.writeChar(' {');
userFile.writeChars(userName);
userFile.writeChars(" , ");
userFile.writeChars(passVerifier.toString());
userFile.writeChars(" , ");
userFile.writeChars(salt.toString());
userFile.writeChar(' ');
succeeded = 1;
retry = false;
} catch (IOException addUserE) {
messageText.setText("Error writing new user to file: " +
addUserE.getMessage());
succeeded = -1;
} catch (ArithmeticException ae) {
messageText.setText("---Math problem: " + ae.getMessage());
retry = true;
succeeded = -1;
} catch (Exception e) {
messageText.setText("add user problem: " + e.getMessage());
succeeded = -1;
}
} while(retry);
return(succeeded);
} // end add user

// Variables declaration
private javax.swing.JPanel bgPanel;
private javax.swing.JScrollPane messageSPane;
private javax.swing.JButton exitButton;
private javax.swing.JButton listButton;
private javax.swing.JLabel messageLabel;
private javax.swing.JPasswordField passField;
private javax.swing.JTextField unText;
private javax.swing.JLabel passLabel;
private javax.swing.JButton addUserButton;
private javax.swing.JLabel unLabel;
private static javax.swing.JTextArea messageText;
Appendix B. SHAInterleave.java

/**
 * SHAInterleave.java
 */
package fileAccessManager;
import java.math.*;
import java.security.*;
public class SHAInterleave {
    /** Creates a new instance of SHAInterleave */
    public SHAInterleave() {
    }
    /**
     * interleave returns a 320 bit BigInteger. The input is divided into two
     * byte arrays by first removing leading 0 bytes, then checking to see if
     * the resulting byte array has an even or odd number of bytes. If there
     * are an odd number of bytes the first byte is also removed. Then the
     * array is divided into two smaller byte arrays, The first contains the even
     * numbered bytes and the second the odd numbered bytes. Two new arrays
     * are then produced by calculating the SHA hash of the first two arrays.
     * these new arrays are then put back together alternating even and odd bytes
     * this final 40 byte array is then converted to a BigInteger and that value
     * is returned.
     */
    public static BigInteger interleave(BigInteger input) throws Exception {
        byte[] t = input.toByteArray();
        int start = 0;
        int i = 0;
        int stop = t.length;
        if (t[0] == 0) {
            start = 1;
            stop = t.length - 1;
        }
        if (t.length % 2 == 1) {
            stop--;
            start++;
        }
        byte[] e = new byte[stop/2];
        byte[] f = new byte[stop/2];
        for (i = 0; i < stop; i++) {
            if (((i % 2 == 0) || (i == 0)) {//even numbered bytes
                if (i == 0)
                    e[0] = t[start];
                else e[i/2] = t[start];
            }
            if (((i % 2 == 1) || (i == 1)) {//odd numbered bytes
                if (i == 1)
                    f[0] = t[start];
                else f[i/2] = t[start];
            }
        }
    }
}
```java
start++;

MessageDigest md = MessageDigest.getInstance("SHA");
md.update(c);
byte[] g = md.digest();
md.reset();
md.update(f);
byte[] h = md.digest();
// int length = e.length + f.length;
int length = 40;
byte[] result = new byte[length];
int j = 0;
for (i = 0; i < length; i = i + 2) { // put it back together again
    result[i] = g[j];
    result[i+1] = h[j];
    j++;
}
// to convert to a BigInteger that is positive add an extra 0 byte to the
// front of result[]
byte posResult[] = new byte[length + 1];
posResult[0] = (byte)0;
for (int z = 0; z < length; z++)
    posResult[z+1] = result[z];
BigInteger returnResult = new BigInteger(result);
return (returnResult);
```
Appendix C. SRPServer.java

/*
 * SRPServer.java
 */
package fileAccessManager;
import java.io.*;
import java.net.*;
import java.security.*;
import javax.net.*;
import java.math.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.Key.*;
import java.security.spec.*;
import java.util.*;
public class SRPServer extends Thread {
    public final static int portNum = 50505;
    private static Socket mysock;
    public static BigInteger N;
    public static BigInteger G;
    public static BigInteger salt;
    public static BigInteger passVerifier; /** v */
    public static String U;
    public static BigInteger b;
    public static BigInteger B;
    public static BigInteger A;
    public static BigInteger S;
    public static BigInteger clientM;
    public static BigInteger serverM;
    public static BigInteger M;
    public static byte uO;
    /** hash from Client compare to serverM */
    /** H(A XOR H(G)|H(U)|salt|A|B|K) */
    /** first 32 bit of SHA of B MSB first (4 bytes) */
    private static int bLength = 32;
    private static RandomAccessFile userFile;
    private static boolean noN = true;
    private static PrintWriter pw;
    private static InputStreamReader isr;
    private static BufferedReader br;
    private static ServerSocket mysocket;
    public static RandomAccessFile accessFile;

    /** Creates a new instance of SRPServer */
    public SRPServer(Socket s) {
        mysock = s;
    }
    /** no arguments for main */
public static void main(String[] args) throws Exception {
    //get the values for N and G from the userFile
    try {
        userFile = new RandomAccessFile("userList.txt", "r");
        accessFile = new RandomAccessFile("access.txt", "rw");
    } catch (FileNotFoundException fnfe) {
        System.exit(0);
    }
    getNandG();
    if (noN == true) {
        System.out.println("Please run userManager to create N and G");
        System.exit(0);
    }
    if (N.compareTo(BigInteger.ZERO) == 0) {
        System.out.println("Please run userManager to create N and G");
        System.exit(0);
    }
    mysocket = new ServerSocket(portNum);
    System.out.println(myTime.getTimeInMillis());
    System.out.println("Waiting for Client to connect");
    while (true) {
        new SRPServer(mysocket.accept()).start();
    }
}
public void run() {
    try {
        pw = new PrintWriter(mysock.getOutputStream(), true);
        isr = new InputStreamReader(mysock.getInputStream());
        br = new BufferedReader(isr);
        U = br.readLine(); //get userName
        if (getUserlnfo() != -1) {
            // this is a valid username so continue the handshake process
            pw.println(salt);
            pw.println(N);
            pw.println(G);
            boolean Kerror = false; //if calcK produces an error restart auth
            do {
                if (Kerror == true) System.out.println("Trying again");
                Kerror = false;
                String fromClient = new String(br.readLine()); //read A or abort
                if (fromClient.startsWith("abort")) {
                    System.out.println("Client aborted");
                    myExit();
                    return;
                }
                A = new BigInteger(fromClient);
                // if A % N == 0 then abort the authentication
                if (A.mod(N).equals(BigInteger.ZERO)) {
                    System.out.println("Abort @ A % N");
                    pw.println("abort @ A % N");
                }
            }
        }
    }
}
myExit();
return;
}
if (calcB() == -1) {
    System.out.println("Abort @ B");
pw.println("abort @ B");
myExit();
return;
}
pw.println(B);
if (calcK() == -1) {
    System.out.println("Error @ K");
pw.println("kerror");
    Kerror = true;
} else if (Kerror == false) {
    fromClient = new String(br.readLine());
    if (fromClient.startsWith("kerror")) {
        System.out.println("Client Kerror");
        Kerror = true;
    } else if (Kerror == false) {
        if (fromClient.startsWith("abort")) {
            System.out.println("Client aborted");
            myExit();
            return;
        }
        clientM = new BigInteger(fromClient);
    } while (Kerror == true);

    // Server sends H(A | M | K) after verifying that clientM == serverM
    if (calcM() == -1) {
        System.out.println("Abort @ serverM");
pw.println("abort @ serverM");
myExit();
return;
    }
    if (clientM.compareTo(serverM) != 0) {
        System.out.println("invalid login attempt by user: " + U);
pw.println("abort @ compare M");
myExit();
return;
    }
try {
    MessageDigest md = MessageDigest.getInstance("SHA");
    String hashString = new String(A.toString());
    hashString.concat(clientM.toString());
    hashString.concat(K.toString());
    md.update(hashString.getBytes());
    byte tempArray[] = new byte[20];
byte posTemp[] = new byte[21];
posTemp[0] = (byte)0;
tempArray = md.digest();
for (int z = 0; z < 20; z++)
    posTemp[z+1] = tempArray[z];
M = new BigInteger(posTemp);
} catch (Exception hashE) {
    System.out.println("Calculating M failed");
    pw.println("abort");
    myExit();
    return;
}
pw.println(M);
System.out.println(U + " Authentication complete");
handleFileRequests();
}
else {
    System.out.println("Invalid Username : " + U);
    pw.println("Invalid user name");
}
//close out everything because exchange is done
} catch (Throwable e) {
    System.out.println("Error " + e.getMessage());
    e.printStackTrace();
}
myExit();
}//end run

/** myExit() closes out the printwriter, bufferedreader, inputstreamreader * and socket */
private static void myExit() {
    try {
        pw.close();
        br.close();
        isr.close();
        mysock.close();
    } catch (Throwable e) {
        e.printStackTrace();
    }
}//end myExit()

/** getNandG() reads the values for N and G from the file "userlist.txt" * it sets the global value noN to false if N or G cannot be read correctly * or do not exist. */
public static void getNandG() {
    int spot = 0;
    int i = 0;
    char myChar;
    try { //File has "N value G value" followed by "{username, passverifier, //salt}" for each user separated by a space
        } catch (IOException e) {
            System.out.println("I/O Error: " + e.getMessage());
        }
    } catch (Throwable e) {
        System.out.println("Error " + e.getMessage());
        e.printStackTrace();
    }
    return;
}
//end getNandG()
userFile.seek(spot);
myChar = userFile.readChar();
if (myChar == 'N') {
    spot += 4;
    userFile.seek(spot);
    StringBuffer nsb = new StringBuffer();
    myChar = userFile.readChar();
    while (myChar != ' ') {
        nsb.append(myChar);
        myChar = userFile.readChar();
    }
    N = new BigInteger(nsb.toString());
}
myChar = userFile.readChar();
if (myChar == 'G') {
    myChar = userFile.readChar();
    StringBuffer gsb = new StringBuffer();
    myChar = userFile.readChar();
    while (myChar != ' ') {
        gsb.append(myChar);
        myChar = userFile.readChar();
    }
    G = new BigInteger(gsb.toString());
    noN = false;
}
} catch (Exception raf) {
    System.out.println("Errors reading userList.txt: " + raf.getMessage());
    noN = false;
}
} //end getNandG()

/** getUserInfo() returns 0 if the values for salt and passVerifier are
* read correctly from "userlist.txt". returns -1 if an error occurs or
* the username supplied by the client does not exist in "userlist.txt"
*/
public static int getUserInfo() {
    int succeeded = 0;
    char tempChar;
    StringBuffer unBuf;
    StringBuffer saltBuf = new StringBuffer();
    StringBuffer pvBuf = new StringBuffer();
    boolean gotInfo = false;
    try {
        userFile.seek(0);
        while(gotInfo == false) {
            unBuf = new StringBuffer();
            while (userFile.readChar() != '{'); // read until the first '{
                tempChar = userFile.readChar();
            if (unCharAt(0) == tempChar) { // they start the same
                unBuf.append(tempChar);
                tempChar = userFile.readChar();
                while(tempChar != ',') {
                }
unBuf.append(tempChar);
tempChar = userFile.readChar();
} //get full UN now
if (U.compareTo(unBuf.toString()) == 0) {
    //they are the same name now get pv and salt
    tempChar = userFile.readChar(); //read ",
    tempChar = userFile.readChar(); //read " 
    while(tempChar != ',') {
        pvBuf.append(tempChar);
        tempChar = userFile.readChar();
    } //got full pv
    tempChar = userFile.readChar();
tempChar = userFile.readChar();
    while(tempChar != '}') {
        saltBuf.append(tempChar);
        tempChar = userFile.readChar();
    } //got full salt
    passVerifier = new BigInteger(pvBuf.toString());
salt = new BigInteger(saltBuf.toString());
gotInfo = true;
} //got pv and salt
} //first char matched
} //end while ! gotInfo
} catch (IOException raf) {
    System.out.println("Errors reading userList.txt: " + raf.getMessage());
succeeded = -1; //either something went wrong or the user wasn't there
}
return (succeeded);
} //end getUserInfo()

/**
calcB returns 0 if the value for B was calculated correctly otherwise
* returns -1 if any error occurred.
* B = (v + G^b) % N
*/
public static int calcB() {
    int succeeded = 0;
    int bits = bLength * 8 -1;
    byte tempB[] = new byte[bLength];
    byte byteB[] = new byte[bLength+1];
    try {
        SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
        random.nextBytes(tempB);
        byteB[0] = (byte)0;
        for (int i = 0; i < bLength; i++)
            byteB[i + 1] = tempB[i];
        b = new BigInteger(byteB);
        B = G.modPow(b, N);
        B = B.add(passVerifier).mod(N);
    } catch (Exception e) {
        succeeded = -1;
    }
    return succeeded;
/** calcK returns 0 if the values for K and S are calculated correctly
 * otherwise returns -1 if an error occurred
 * S = (A * v^u) ^ b % N
 * K = SHAInterleave(S)
 */
public static int calcK() {
    int succeeded = 0;
    // calculate u = first 32 bits of SHA(B) MSB first
    u = new byte[5];
    u[0] = (byte)0;
    try {
        MessageDigest md = MessageDigest.getInstance("SHA");
        md.update(B.toByteArray());
        byte temp[] = md.digest();
        for (int i = 0; i < 4; i++)
            u[i+l] = temp[i];
        BigInteger bigU = new BigInteger(u);
        // now calculate S
        BigInteger number = passVerifier.modPow(bigU, N);
        number = number.multiply(A);
        S = number.modPow(b, N);
        K = SHAInterleave.interleave(S);
    } catch (ArithmeticException mathe) {
        System.out.println("Math error in calcK: " + mathe.getMessage());
        succeeded = -1;
    } catch (Exception e) {
        System.out.println("Error in calcK: " + e.getMessage());
        succeeded = -1;
    }
    return succeeded;
}

/** calcM returns 0 if the value of serverM was calculated correctly
 * returns -1 if an error occurred
 * M = H( (H(N) xor H(G) | H(U) | salt | A | B | K)
 */
public static int calcM() {
    int succeeded = 0;
    try {
        MessageDigest md = MessageDigest.getInstance("SHA");
        StringBuffer hashString = new StringBuffer();
        md.update(N.toByteArray());
        byte tempArray[] = new byte[20];
        byte posTemp[] = new byte[21];
        int z;
        posTemp[0] = (byte)0;
        tempArray = md.digest();
        for (z = 0; z < 20; z++)
            posTemp[z+1] = tempArray[z];
        BigInteger tempHash = new BigInteger(posTemp);
md.update(G.toByteArray());
posTemp[0] = (byte)0;
tempArray = md.digest();
for (z = 0; z < 20; z++)
    posTemp[z+1] = tempArray[z];
tempHash.xor(new BigInteger(posTemp));
hashString.append(tempHash.toString());
md.update(U.getBytes());
hashString.append(new BigInteger(md.digest()).toString());
hashString.append(salt.toString());
hashString.append(A.toString());
hashString.append(B.toString());
hashString.append(K.toString());
md.update(new String(hashString.toString()).getBytes());
posTemp[0] = (byte)0;
tempArray = md.digest();
for (z = 0; z < 20; z++)
    posTemp[z+1] = tempArray[z];
serverM = new BigInteger(posTemp);
} catch (Exception e) {
    succeeded = -1;
}
return succeeded;
}//end calcM()

public static void handleFileRequests() {
    String request = "exit";
    char myChar;
    BigInteger nextFileID;
    /* look to see if any fileIDs exist in the accessFile. If they do then
     * nextFileID = the last ID + 1. If the accessFile is empty then
     * nextFileID = 1.
     * File info is stored:
     * {fileID, filename, key, owner, userlist}
     */
    try {
        long spot = 0;
        accessFile.seek(spot);
        if (accessFile.length() == 0)
            nextFileID = new BigInteger("1");
        else if ((spot = accessFile.length() - 1) == 0)
            nextFileID = new BigInteger("1");
        else {
            accessFile.seek(spot-1);
            while ((myChar = accessFile.readChar()) != '{') {
                spot--;
                accessFile.seek(spot);
            } //location of last " {
            StringBuffer numSB = new StringBuffer();
            myChar = accessFile.readChar();
            while(myChar!='}') {
                numSB.append(myChar);
myChar = accessFile.readChar();
}
nextFileID = new BigInteger(numSB.toString());
nextFileID = nextFileID.add(BigInteger.ONE);
//fileIDs already existed

int i = 0;
int num = 0;
StringBuffer users;
String filename;
String owner = new String(U);
BigInteger filekey;
String fileID;
do {
    users = new StringBuffer();
    request = new String(receive());
    if (request.compareTo("GetKey") == 0) {
        filename = new String(receive());
        num = Integer.parseInt(receive());
        for (i = 0; i < num - 1; i++) {
            users.append(receive());
            users.append(" ");
        }
    }
    users.append(receive());
    // compute key and send it to the client
    filekey = generateFileKeyQ;
    if (filekey.compareTo(BigInteger.ZERO) != 0) {
        spot = accessFile.length();
        accessFile.seek(spot);
        accessFile.writeChar(\x27{\x27);
        accessFile.writeChars(nextFileID.toString());
        accessFile.writeChars(\x27, \x27);
        accessFile.writeChars(filename);
        accessFile.writeChars(\x27, \x27);
        accessFile.writeChars(filekey.toString());
        accessFile.writeChars(\x27, \x27);
        accessFile.writeChars(owner);
        accessFile.writeChars(\x27, \x27);
        accessFile.writeChars(users.toString());
        accessFile.writeChars(\x27}\x27);
        if (send(filekey.toString()) == -1)
            System.out.println("Error sending");
        if (send(nextFileID.toString()) == -1)
            System.out.println("Error sending");
    } else {
        if (send("error") == -1)
            System.out.println("Error sending");
    }
} //end Get Key request

//getFileInfo request
//fileID received fileID, filename, filekey and users returned
if (request.compareTo("Get File Info") == 0) {
    boolean fileFound = false;
    accessFile.seek(0);
    StringBuffer myFileBuf;
    StringBuffer filenameBuf = new StringBuffer();
    StringBuffer keyBuf = new StringBuffer();
    StringBuffer ownerBuf = new StringBuffer();
    StringBuffer listBuf = new StringBuffer();
    fileID = new String(receive());
    //requested fileID is larger than the nextFileID value
    if (nextFileID.compareTo(new BigInteger(fileID)) <= 0) {
        fileFound = true;
        if (send("FileID not found") == -1)
            System.out.println("Error sending");
    }
    if (accessFile.length() < 10) {//not large enough
        fileFound = false;
        if (send("FileID not found") == -1)
            System.out.println("Error sending");
    }
    while (fileFound == false) {
        myFileBuf = new StringBuffer();
        try {
            myChar = accessFile.readChar();
            while (myChar != '{') {
                System.out.print(myChar);
                myChar = accessFile.readChar();
            }
            System.out.println();
        } catch (IOException end) {
            if (send("FileID not found") == -1)
                System.out.println("Error sending");
            fileFound = false;
            break;
        }
        myChar = accessFile.readChar();
        if (fileID.charAt(0) == myChar) { //first digit same
            myFileBuf.append(myChar);
            myChar = accessFile.readChar();
            while (myChar != ',') {
                myFileBuf.append(myChar);
                myChar = accessFile.readChar();
            }
            if (fileID.compareTo(myFileBuf.toString()) == 0) {
                myChar = accessFile.readChar(); //read next " 
                myChar = accessFile.readChar(); //read filename
                while (myChar != ',') {
                    filenameBuf.append(myChar);
                    myChar = accessFile.readChar();
                } // got file name
                myChar = accessFile.readChar(); // read " 
            } else { //next
                fileFound = true;
            }
        }
    }
} else {
    System.out.println("Wrong request");
}
myChar = accessFile.readChar(); // read key
while(myChar != ',') {
    keyBuf.append(myChar);
    myChar = accessFile.readChar();
}// get key
myChar = accessFile.readChar(); // read
myChar = accessFile.readChar(); // read owner
while(myChar != ',') {
    ownerBuf.append(myChar);
    myChar = accessFile.readChar();
}// get owner
myChar = accessFile.readChar(); // read
myChar = accessFile.readChar(); // read list
while(myChar != '}') {
    listBuf.append(myChar);
    myChar = accessFile.readChar();
}// get list
fileFound = true;
if (listBuf.toString().indexOf(U) != -1) {
    if (send(fileID) == -1)
        System.out.println("error sending");
    if (send(filenameBuf.toString()) == -1)
        System.out.println("error sending");
    if (send(keyBuf.toString()) == -1)
        System.out.println("error sending");
    if (send(listBuf.toString()) == -1)
        System.out.println("error sending");
} else if (send("access denied") == -1)
    System.out.println("error sending to client");
}// end ID match
}// end first digit of ID match
}// end fileFound
}// end getFileInfo request
} while(request.compareTo("exit") != 0);
} catch (EOFException eof) {
    if (send("End of File error") == -1)
        System.out.println("Error sending");
    eof.printStackTrace();
} catch (Exception e) {
    System.out.println("Error in handling requests " + e.toString());
    if (send("Error" + e.getMessage()) == -1)
        System.out.println("Error sending");
}
}// end handleFileRequests

public static BigInteger generateFileKey() {
    BigInteger biKey = BigInteger.ZERO;
    try {
        KeyGenerator kg = KeyGenerator.getInstance("DES");
        SecretKey key = kg.generateKey();
        Class spec = Class.forName("javax.crypto.spec.DESKeySpec");
        SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
        }
DESKeySpec ks = (DESKeySpec)skf.getKeySpec(key, spec);
key = skf.generateSecret((KeySpec)ks);
biKey = new BigInteger(key.getEncoded());
} catch (NoSuchAlgorithmException alg) {
    biKey = BigInteger.ZERO;
} catch (Exception e) {
    biKey = BigInteger.ZERO;
}
return biKey;
} // end getKey

// send and receive messages could be encrypted at a later date
public static int send(String msg) {
    int succeeded = 0;
    try {
        pw.println(msg);
    } catch (Exception e) {
        System.out.println("Error sending to client " + e.getMessage());
        succeeded = -1;
    }
    return succeeded;
} // end send

public static String receive() {
    String text;
    try {
        text = br.readLine();
    } catch (IOException e) {
        text = new String("ERROR");
    }
    return text;
} // end receive
} // end SRPServer class
Appendix D. SRPClient.java

```java
package fileAccessManager;
import java.io.*;
import java.net.*;
import java.security.*;
import javax.net.*;
import java.math.*;
import java.util.*;
public class SRPClient {
    private static final int serverPort = 50505;
    public static Socket mysocket;
    public static PrintWriter pw;
    public static BufferedReader br;
    public static InputStreamReader isr;
    public static BigInteger N;
    public static BigInteger G;
    private static BigInteger salt;
    private static BigInteger passVerifier;
    private static BigInteger a;
    private static int aLength = 32;
    private static BigInteger A;
    private static BigInteger B;
    private static BigInteger x;
    private static String pass;
    private static String userName;
    private static BigInteger S;
    private static BigInteger k;
    private static BigInteger M;
    private static byte uO;
    public static String serverName;
    public static ClientGUI authWindow;

    public SRPClient(ClientGUI parent) {
        authWindow = parent;
    }

    public static int Authenticate(String username, String password, String server) {
        userName = new String(username);
        pass = new String(password);
        serverName = new String(server);
        int succeeded = 0;
        try {
            mysocket = new java.net.Socket(serverName, serverPort);
            pw = new java.io.PrintWriter(mysocket.getOutputStream(), true);
            isr = new InputStreamReader(mysocket.getInputStream());
```
br = new java.io.BufferedReader(isr);
authWindow.message = "connecting to server";
pw.println(userName); //send message
// if the username is valid the server sends salt, N and G
// otherwise it sends "Invalid user name"
String fromServer = new String(br.readLine());
if (fromServer.startsWith("Invalid")) {
    authWindow.message = "Invalid user name";
    myExit();
    succeeded = -1;
    return succeeded;
}
salt = new BigInteger(fromServer);
N = new BigInteger(br.readLine());
G = new BigInteger(br.readLine());
//calculate A and send it to Server
boolean kerror = false;
do {
    if (kerror == true) System.out.println("Trying again");
    kerror = false;
    if (calcA == -1) {
        pw.println("abort");
        myExit();
        succeeded = -1;
        return succeeded;
    }
    pw.println(A);
    if (calcX == -1) {
        authWindow.message = "Aborted @ X";
        pw.println("abort");
        myExit();
        succeeded = -1;
        return succeeded;
    }
}
//read in B or error message from Server
fromServer = new String(br.readLine());
if (fromServer.startsWith("abort")) {
    authWindow.message = "Server is aborting: " + fromServer;
    myExit();
    succeeded = -1;
    return succeeded;
}
B = new BigInteger(fromServer);
// if B % N == 0 abort authentication
if (B.mod(N).compareTo(BigInteger.ZERO) == 0) {
    authWindow.message = "Aborted @ B%N";
    pw.println("abort");
    succeeded = -1;
    myExit();
    return succeeded;
}
// calculate S and K and M  Send M to server
if (calcK() == -1) {
    authWindow.message = "error @ K";
    kerror = true;
    try {
        fromServer = new String(br.readLine());
        if (fromServer.startsWith("kerror")) {
            authWindow.message = "kerror from server";
            System.out.println("Kerror from server");
        }
        if (!fromServer.startsWith("kerror"))
            pw.println("kerror");
        fromServer = new String(br.readLine());
        if (fromServer.startsWith("abort")) {
            succeeded = -1;
            System.out.println("Server aborted at K: " + fromServer);
            myExit();
            return succeeded;
        }
    } catch (IOException err) {
    }
} while (kerror == true);
if (calcM() == -1) {
    authWindow.message = "Aborted @ M"
    pw.println("abort");
    succeeded = -1;
    myExit();
    return succeeded;
}

pw.println(M);
fromServer = new String(br.readLine());
if (fromServer.startsWith("abort")) {
    System.out.println("Server is aborting");
    authWindow.message = "Invalid username or password"
    myExit();
    succeeded = -1;
    return succeeded;
} 
succeeded = 0;
} catch (Exception e) {
    authWindow.message = "Error: " + e.getMessage();
    succeeded = -1;
}
return succeeded;

/** myExit closes out the printwriter, buffered reader, input stream reader
 * and socket before exiting the program */
public static void myExit() {
    // close out all connectionsstreamssockets
    try {

**calcA** returns 0 if the value of A was calculated correctly
* returns -1 if an error occurred
* A = G^a %N
*/

```java
private static int calcA() {
    int succeeded = 0;
    int bits = aLength * 8 - 1;
    byte tempA[] = new byte[aLength];
    byte byteA[] = new byte[aLength + 1];
    try {
        SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
        random.nextBytes(tempA);
        byteA[0] = (byte) 0;
        for (int i = 0; i < aLength; i++)
            byteA[i + 1] = tempA[i];
        a = new BigInteger(byteA);
        A = G.modPow(a, N);
    } catch (Exception e) {
        succeeded = -1;
    }
    return succeeded;
}
```

**calcX** returns 0 if the value of X was calculated and
* returns -1 if an error occurred
* X = SHA(salt | SHA(userName | "." | pass))
*/

```java
private static int calcX() {
    int succeeded = 0;
    try {
        MessageDigest md = MessageDigest.getInstance("SHA");
        String userPass = new String(userName);
        userPass = userPass.concat(":");
        userPass = userPass.concat(pass);
        md.update(userPass.getBytes());
        byte temp[] = md.digest();
        String saltStr = new String(salt.toString());
        saltStr = saltStr.concat(new String(temp));
        md.update(saltStr.getBytes());
        byte smallX[] = new byte[20];
        byte posX[] = new byte[21];
        posX[0] = (byte) 0;
        smallX = md.digest();
        for (int i = 0; i < 20; i++)
```
posX[i+1] = smallX[i];
x = new BigInteger(posX);
} catch (Exception e) {
succeeded = -1;
}
return succeeded;
}  // end calcX()

/** CalcK returns 0 if the values for S and K were successfully calculated
 * returns -1 if there was an error in the calculations
 * S = (B - G^x)^(a + u * x) % N
 * K = SHAInterleave(S)
 */
private static int calcK() {
    int succeeded = 0;
    // calculate u = first 32 bits of SHA(B) MSB first
    u = new byte[5];
    u[0] = (byte)0;
    try {
        MessageDigest md = MessageDigest.getInstance("SHA");
        md.update(B.toByteArray());
        byte temp[] = md.digest();
        for (int i = 0; i < 4; i++)
            u[i+1] = temp[i];
        BigInteger U = new BigInteger(u);
        // now calculate S
        BigInteger number = B;
        BigInteger pv = G.modPow(x,N);
        number = number.subtract(pv);
        BigInteger exponent = U;
        BigInteger exponentA = exponent.multiply(x);
        BigInteger exponentB = exponentA.add(a);
        S = number.modPow(exponentB, N);
        K = SHAInterleave.interleave(S);
    } catch (ArithmeticException mathe) {
        succeeded = -1;
    } catch (Exception e) {
        succeeded = -1;
    }
    return succeeded;
}  // end calcK()

/** calcM returns 0 if the value of M was calculated successfully and
 * returns -1 if an error occurred.
 * M = H( H(N) xor H(G) | H(U) | salt | A | B | K)
 */
private static int calcM() {
    int succeeded = 0;
    try {
        MessageDigest md = MessageDigest.getInstance("SHA");
        StringBuffer hashString = new StringBuffer();
        md.update(N.toByteArray());
        hashString.append(md.digest());
        String salt = "salt";
        String a = "A";
        String b = "B";
        String k = "K";
        hashString.append(salt + a + b + k);
        String result = md.digest(hashString.toString().getBytes());
        BigInteger number = new BigInteger(result, 16);
        BigInteger pv = G.modPow(x,N);
        number = number.subtract(pv);
        BigInteger exponent = U;
        BigInteger exponentA = exponent.multiply(x);
        BigInteger exponentB = exponentA.add(a);
        S = number.modPow(exponentB, N);
        K = SHAInterleave.interleave(S);
    } catch (ArithmeticException mathe) {
        succeeded = -1;
    } catch (Exception e) {
        succeeded = -1;
    }
    return succeeded;
}  // end calcM()
byte tempArray[] = new byte[20];
byte posTemp[] = new byte[21];
int z;
posTemp[0] = (byte)0;
tempArray = md.digest();
for (z = 0; z < 20; z++)
    posTemp[z+1] = tempArray[z];
BigInteger tempHash = new BigInteger(posTemp);
md.update(G.toByteArray());
posTemp[0] = (byte)0;
tempArray = md.digest();
for (z = 0; z < 20; z++)
    posTemp[z+1] = tempArray[z];
tempHash.xor(new BigInteger(posTemp));
hashString.append(tempHash.toString());
md.update(userName.getBytes());
hashString.append(new BigInteger(md.digest()).toString());
hashString.append(salt.toString());
hashString.append(A.toString());
hashString.append(B.toString());
hashString.append(K.toString());
md.update(new String(hashString.toString()).getBytes());
posTemp[0] = (byte)0;
tempArray = md.digest();
for (z = 0; z < 20; z++)
    posTemp[z+1] = tempArray[z];
M = new BigInteger(posTemp);
}

return succeeded;

//end calcMO

public static int send(String msg) {
    int succeeded = 0;
    try {
        pw.println(msg);
    } catch (Exception e) {
        succeeded = -1;
    }
    return succeeded;
}

public static String receive() {
    String text;
    try {
        text = br.readLine();
    } catch (IOException ioe) {
        text = new String("ERROR");
    }
    return text;
}

//end SPRClient class
Appendix E. ClientGUI.java

/*
 * ClientGUI.java
 */
package fileAccessManager;
import javax.swing.*;
import java.awt.*;
import java.util.*;
public class ClientGUI extends javax.swing.JFrame {
    public static String userName;
    public static String pass;
    public static String serverName;
    public static SRPClient client;
    public static Object message;

    /** Creates new form ClientGUI */
    public ClientGUI() {
        client = new SRPCClient(this);
        initComponents();
    }

    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;
        background.Panel = new javax.swing.JPanel();
        userLabel = new javax.swing.JLabel();
        usernameText = new javax.swing.JTextField();
        passLabel = new javax.swing.JLabel();
        titleLabel = new javax.swing.JLabel();
        passwordField = new javax.swing.JPasswordField();
        okButton = new javax.swing.JButton();
        cancelButton = new javax.swing.JButton();
        getContentPane().setLayout(new java.awt.GridBagLayout());
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });
        backgroundPanel.setLayout(new java.awt.GridBagLayout());
        backgroundPanel.setPreferredSize(new java.awt.Dimension(300, 200));
        userLabel.setText("Username");
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.gridwidth = 1;
        gridBagConstraints.gridheight = 1;
        gridBagConstraints.weightx = 1.
        gridBagConstraints.weighty = 1.
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        backgroundPanel.add(userLabel, gridBagConstraints);
        gridBagConstraints.gridx = 1;
        gridBagConstraints.weightx = 1.
        gridBagConstraints.weighty = 1.
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        backgroundPanel.add(usernameText, gridBagConstraints);
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.weightx = 1.
        gridBagConstraints.weighty = 1.
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        backgroundPanel.add(passLabel, gridBagConstraints);
        gridBagConstraints.gridx = 1;
        gridBagConstraints.weightx = 1.
        gridBagConstraints.weighty = 1.
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        backgroundPanel.add(passwordField, gridBagConstraints);
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.weightx = 1.
        gridBagConstraints.weighty = 1.
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        backgroundPanel.add(titleLabel, gridBagConstraints);
        gridBagConstraints.gridx = 1;
        gridBagConstraints.weightx = 1.
        gridBagConstraints.weighty = 1.
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        backgroundPanel.add(okButton, gridBagConstraints);
        gridBagConstraints.gridx = 2;
        gridBagConstraints.weightx = 1.
        gridBagConstraints.weighty = 1.
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        backgroundPanel.add(cancelButton, gridBagConstraints);
        getContentPane().add(backgroundPanel);  
    }

    backgroundPanel.setMinimumSize(new java.awt.Dimension(300, 200));
    backgroundPanel.setPreferredSize(new java.awt.Dimension(300, 200));
    userLabel.setLabelText("Username");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 0;
    gridBagConstraints.gridwidth = 1;
    gridBagConstraints.gridheight = 1;
    gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
    gridBagConstraints.weightx = 1.
    gridBagConstraints.weighty = 1.
    gridBagConstraints.weightwidth = 1.
    gridBagConstraints.weightheight = 1.
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    backgroundPanel.setLayout(gridBagLayout);
    backgroundPanel.add(usernameText, gridBagConstraints);
    backgroundPanel.add(passwordField, gridBagConstraints);
    backgroundPanel.add(okButton, gridBagConstraints);
    backgroundPanel.add(cancelButton, gridBagConstraints);
}
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(userLabel, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(usernameText, gridBagConstraints);

passLabel.setText("Password");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(passLabel, gridBagConstraints);

titleLabel.setText("Please Enter Username and Password");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(titleLabel, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 2;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(passwordField, gridBagConstraints);

okButton.setText("OK");
okButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    okButtonActionPerformed(evt);
  }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(okButton, gridBagConstraints);

cancelButton.setText("Cancel");
cancelButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    cancelButtonActionPerformed(evt);
  }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(cancelfiutton, gridBagConstraints);

cancelfiutton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        client.myExit();
        System.exit(0);
    }
});

private void cancelButtonActionPerformed(java.awt.event.ActionEvent evt) {
    userName = usernameText.getText();
    pass = new String(passwordField.getPassword());
    if (client.Authenticate(userName, pass, serverName) == -1) {
        String title = "Error during Authentication";
        int messageType = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(this, message, title, messageType);
        System.exit(0);
    } else {
        new FileIDgui(this, client, userName).show();
    }
}

private void exitForm(java.awt.event.WindowEvent evt) {
    client.pw.println("exit");
    client.myExit();
    System.exit(0);
}

/**
 * @param args[0] - server name
 */
public static void main(String[] args) {
    serverName = args[0];
    message = new StringBuffer();
    new ClientGUI().show();
}

// Variables declaration
private javax.swing.JLabel titleLabel;
private javax.swing.JPanel backgroundPanel;
private javax.swing.JButton cancelButton;
private javax.swing.JButton okButton;
private javax.swing.JLabel userLabel;
private javax.swing.JLabel passLabel;
private javax.swing.JTextField usernameText;
private javax.swing.JPasswordField passwordField;
private javax.swing.JPasswordField passwordField;
Appendix F. FileIDgui.java

```java
/*
 * FileIDgui.java
 */
package fileAccessManager;
import javax.swing.*;
import java.math.*;
import java.io.*;
import javax.crypto.*;
import java.util.*;
public class FileIDgui extends javax.swing.JFrame {
    public static String fileName;
    public static String fileID;
    public static BigInteger key;
    public static String keyS;
    public static String accessList;
    public static String messageS;
    private static FAMcrypto crypto;
    private static SRPClient client;

    /** Creates new form FileIDgui */
    public FileIDgui(ClientGUI parent, SRPClient Userclient, String user) {
        client = Userclient;
        parent.disposeQ;
        username = new String(user);
        crypto = new FAMcrypto(this);
        initComponents();
    }
    public FileIDgui() {
        initComponents();
    }
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;
        backgroundPanel = new javax.swing.JPanel();
        IDlabel = new javax.swing.JLabel();
        fileIDText = new javax.swing.JTextField();
        keyLabel = new javax.swing.JLabel();
        keyText = new javax.swing.JTextField();
        infoButton = new javax.swing.JButton();
        keyButton = new javax.swing.JButton();
        encryptButton = new javax.swing.JButton();
        decryptButton = new javax.swing.JButton();
        fileNameLabel = new javax.swing.JLabel();
        fileNameText = new javax.swing.JTextField();
        //...
    }
}
```
editUsersButton = new javax.swing.JButtonQ;
userScrollPane = new javax.swing.JScrollPaneQ;
userListTA = new javax.swing.JTextAreaQ;
userListTA.setText(username);
exitButton = new javax.swing.JButtonQ;
accessListLabel = new javax.swing.JLabelQ;
clearButton = new javax.swing.JButtonQ;
spacerPanel = new javax.swing.JPanelQ;
spacerPanel1 = new javax.swing.JPanelQ;
spacerPanel2 = new javax.swing.JPanelQ;
spacerPanel3 = new javax.swing.JPanelQ;
messageLabel = new javax.swing.JLabelQ;
messageText = new javax.swing.JTextFieldQ;
getContentPane().setLayout(new java.awt.GridBagLayout());
setTitle("File ID Manager");
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
    }
});

backgroundPanel.setLayout(new java.awt.GridBagLayout());
IDlabel.setText("File ID");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(IDlabel, gridBagConstraints);
fileIDText.setMinimumSize(new java.awt.Dimension(100, 20));
fileIDText.setPreferredSize(new java.awt.Dimension(100, 20));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 2;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(fileIDText, gridBagConstraints);
keyLabel.setText("Key");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(keyLabel, gridBagConstraints);
keyText.setMinimumSize(new java.awt.Dimension(100, 20));
keyText.setPreferredSize(new java.awt.Dimension(100, 20));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 3;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(keyText, gridBagConstraints);

infoButton.setText("Get File Info");
infoButton.setActionCommand("infoButton");
infoButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        infoButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(infoButton, gridBagConstraints);

keyButton.setText("Get Key");
keyButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        keyButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(keyButton, gridBagConstraints);

encryptButton.setText("Encrypt File");
encryptButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        encryptButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 5;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(encryptButton, gridBagConstraints);

decryptButton.setText("Decrypt File");
decryptButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        decryptButtonActionPerformed(evt);
    }
});
gridBagConstraints = new java.awt GridBagConstraints;
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 6;
gridBagConstraints.fill = java.awt GridBagConstraints.BOTH;
backgroundPanel.add(decryptButton, gridBagConstraints);

fileNameLabel.setText("File Name");
gridBagConstraints = new java.awt GridBagConstraints;
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt GridBagConstraints.BOTH;
backgroundPanel.add(fileNameLabel, gridBagConstraints);

fileNameText.setMinimumSize(new java.awt.Dimension(150, 20));
fileNameText.setPreferredSize(new java.awt.Dimension(150, 20));
gridBagConstraints = new java.awt GridBagConstraints;
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.fill = java.awt GridBagConstraints.BOTH;
backgroundPanel.add(fileNameText, gridBagConstraints);

editUsersButton.setText("Edit Access List");
etCursor.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        editUsersButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt GridBagConstraints;
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 5;
gridBagConstraints.fill = java.awt GridBagConstraints.BOTH;
backgroundPanel.add(editUsersButton, gridBagConstraints);

userScrollPane.setMinimumSize(new java.awt.Dimension(100, 100));
userScrollPane.setPreferredSize(new java.awt.Dimension(100, 100));
userListTA.setEditable(false);
userListTA.setMinimumSize(new java.awt.Dimension(100, 100));
userListTA.setPreferredSize(new java.awt Dimension(100, 100));
userScrollPane.setViewportView(userListTA);

gridBagConstraints = new java.awt GridBagConstraints;
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 6;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.gridheight = 4;
gridBagConstraints.insets = new java.awt Insets(1, 1, 1, 1);
backgroundPanel.add(userScrollPane, gridBagConstraints);
exitButton.setText("Exit");
extitButton.addActionListener(new java.awt.event.ActionEvent() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
exitButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 9;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 1;
gridBagConstraints.ipady = 1;
gridBagConstraints.insets = new java.awt.Insets(1, 1, 1, 1);
backgroundPanel.add(exitButton, gridBagConstraints);

accessListLabel.setText("Access List:");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 5;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(accessListLabel, gridBagConstraints);

clearButton.setText("Clear Form");
clearButton.addActionListener(new java.awt.event.ActionEvent() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
clearButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 8;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 1;
gridBagConstraints.ipady = 1;
gridBagConstraints.insets = new java.awt.Insets(1, 1, 1, 1);
backgroundPanel.add(clearButton, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 4;
gridBagConstraints.gridwidth = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(spacerPanel, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 7;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.gridwidth = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(spacerPanel, gridBagConstraints);

spacerPanel1.setPreferredSize(new java.awt.Dimension(50, 50));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 7;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(spacerPanel1, gridBagConstraints);

spacerPanel2.setMinimumSize(new java.awt.Dimension(50, 20));
spacerPanel2.setPreferredSize(new java.awt.Dimension(50, 50));
gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 5;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
backgroundPanel.add(spacerPanel2, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.gridwidth = 4;
gridBagConstraints.gridheight = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 1;
gridBagConstraints.ipady = 1;
gridBagConstraints.insets = new java.awt.Insets(1, 1, 1, 1);
getContentPane().add(backgroundPanel, gridBagConstraints);

messageLabel.setText("Messages:");
gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 5;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 1;
gridBagConstraints.ipady = 1;
gridBagConstraints.insets = new java.awt.Insets(1, 1, 1, 1);
getContentPane().add(messageLabel, gridBagConstraints);

messageText.setText("Welcome to FileAccessManager");
messageText.setMinimumSize(new java.awt.Dimension(100, 20));
messageText.setPreferredSize(new java.awt.Dimension(100, 20));
gridBagConstraints = new java.awt.GridBagConstraints;
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 5;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 1;
gridBagConstraints.ipady = 1;
gridBagConstraints.insets = new java.awt.Insets(1, 1, 1, 1);
getContentPane().add(messageText, gridBagConstraints);
pack();
private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
    fileNameText.setText(" ");
    keyText.setText(" ");
    fileIDText.setText(" ");
    userListTA.setText(username);
    messageText.setText(" ");
}

private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (client.send("exit") == -1) {
        messageText.setText("Error exiting server");
    }
    client.myExit();
    System.exit(0);
}

private void decryptButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String dfileName = new String(fileIDText.getText()) + "enc";
    fileID = new String(fileIDText.getText());
    Object confirmDecrypt = "Decrypt file " + dfileName + "?";
    String confirmDtitle = "Decryption confirmation Dialog";
    int optionType = JOptionPane.YES_NO_OPTION;
    int messageType = JOptionPane.QUESTION_MESSAGE;
    if (JOptionPane.showConfirmDialog(this, confirmDecrypt, confirmDtitle, optionType, messageType) == 0) {
        //Yes was selected
        if (crypto.decrypt(dfileName, key, fileID) == -1) {
            messageText.setText("Error decrypting file");
        } else messageText.setText("File decrypted");
    }
}

private void editUsersButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String temp = userListTA.getText();
    int userNumber = userListTA.getLineCount();
    accessList = new String(temp);
    new UserListEditor(this, accessList).show();
}

private void encryptButtonActionPerformed(java.awt.event.ActionEvent evt) {
    fileName = new String(fileNameText.getText());
    fileID = new String(fileIDText.getText());
    Object confirmEncrypt = "Encrypt file " + fileName + "?";
    String confirmEtitle = "Encryption confirmation Dialog";
    int optionType = JOptionPane.YES_NO_OPTION;
    int messageType = JOptionPane.QUESTION_MESSAGE;
    Object status = "Encryption in process";
    String statusTitle = "In Process";
    if (JOptionPane.showConfirmDialog(this, confirmEncrypt, confirmEtitle, optionType, messageType) == 0) {
        //Encryption in progress
        if (crypto.encrypt(fileName, key, fileID) == -1) {
            messageText.setText("Error encrypting file");
        } else messageText.setText("File encrypted");
    }
}
// Yes was selected
if (crypto.encrypt(fileName, key, filelD) == -1) {
    messageText.setText("Error encrypting file");
} else messageText.setText("File encrypted");
}

private void keyButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String temp = new String(userListTA.getText());
    String fromServer;
    int userNumber = userListTA.getLineCount();
    accessList = new String(temp);
    if (client.send("Get Key") == -1) {
        messageText.setText("Error sending key request");
    }
    if (client.send(fileName.getText()) == -1) {
        messageText.setText("Error sending filename");
    }
    if (client.send(Integer.toString(userNumber)) == -1) {
        messageText.setText("Error sending number of users");
    }
    if (client.send(accessList) == -1) {
        messageText.setText("Error sending accessList");
    }
    try {
        fromServer = new String(client.receive());
        if (fromServer.compareToIgnoreCase("error") == 0) {
            messageText.setText("Error getting key from server");
        } else {
            key = new BigInteger(fromServer);
            keyText.setText("Key obtained");
        }
        fromServer = new String(client.receive());
        if (fromServer.compareToIgnoreCase("error") == 0) {
            messageText.setText("Error getting FileID from server");
        } else {
            fileID = new String(fromServer);
            fileIDText.setText(fileID);
            messageText.setText("Key and fileID assigned");
        }
    } catch (Exception e) {
        messageText.setText("Error: "+ e.getMessage());
    }
}

private void infoButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String fromServer;
    String fileID = fileIDText.getText();
    if (client.send("Get File Info") == -1) {
        messageText.setText("Error asking for file info");
    }
    if (client.send(fileID) == -1) {
messageText.setText("Error sending fileID");

fromServer = new String(client.receive());
if (fromServer.compareTo(fileID) == 0) { // file was found
    fromServer = new String(client.receive());
    if (fromServer.compareTo("ERROR") != 0) fileNameText.setText(fromServer);
    fromServer = new String(client.receive());
    if (fromServer.compareTo("ERROR") != 0) keyText.setText("Key obtained");
    fromServer = new String(client.receive());
    if (fromServer.compareTo("ERROR") != 0) {
        // parse accessList string so that each user is on a separate line
        userListTA.setText("" + fromServer);
        String un;
        int start = 0;
        int stop = fromServer.indexOf(" ");
        while(stop != -1) {
            un = new String(fromServer.substring(start, stop));
            userListTA.append(un + 
" ");
            start = stop + 1;
            stop = fromServer.indexOf(" ", start);
        }
        un = new String(fromServer.substring(start));
        userListTA.append(un);
    } // end parse username list
    messageText.setText("File information retrieved");
} else if (fromServer.compareTo("access denied") == 0) {
    messageText.setText(fromServer);
    fileNameText.setText(""");
    keyText.setText(""");
    fileIDText.setText("" + fromServer);
    userListTA.setText(username);
} else {
    messageText.setText("FileID not found by server");
    fileNameText.setText(""");
    keyText.setText(""");
    fileIDText.setText(""");
    userListTA.setText(username);
}

/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt) {
    if (client.send("exit") != -1) {
        messageText.setText("Error exiting server");
    }
    client.myExit();
    System.exit(0);
}

// Variables declaration
private javax.swing.JPanel spacerPanel3;
private javax.swing.JPanel spacerPanel2;
private javax.swing.JPanel spacerPanel1;
private javax.swing.JLabel IDlabel;
private javax.swing.JPanel spacerPanel;
private javax.swing.JButton encryptButton;
public static javax.swing.JTextArea userListTA;
private javax.swing.JPanel backgroundPanel;
private javax.swing.JTextField fileNameText;
private javax.swing.JLabel accessListLabel;
private javax.swing.JButton keyButton;
private javax.swing.JButton exitButton;
private javax.swing.JButton editUsersButton;
private javax.swing.JLabel messageLabel;
private javax.swing.JTextField keyText;
private javax.swing.JTextField fileIDText;
private javax.swing.JButton clearButton;
private javax.swing.JButton decryptButton;
private javax.swing.JLabel keyLabel;
public static javax.swing.JTextField messageText;
private javax.swing.JLabel fileNameLabel;
private javax.swing.JScrollPane userScrollPane;
private javax.swing.JButton infoButton;
 Appendix G. UserListEditor.java

-package fileAccessManager;
-import java.io.*;
-import java.net.*;
-import java.security.*;
-import java.math.*;
-import java.util.*;
-import javax.swing.*;
-import javax.swing.text.*;
-
-public class UserListEditor extends javax.swing.JFrame {
   
   public static String userList;
   public static String returnUserList;
   public static FileIDgui returnWindow;

   /*
    * UserListEditor.java
    */
   
   /**
    * Creates new form userList
    */
   public UserListEditor(FileIDgui parent, String userListText) {
      returnWindow = parent;
      userList = new String(userListText);
      initComponents();
   }

   private void initComponents() {
      java.awt.GridBagConstraints gridBagConstraints;

      mainPanel = new javax.swing.JPanel();
      userListLabel = new javax.swing.JLabel();
      userNameText = new javax.swing.JTextArea();
      okButton = new javax.swing.JButton();
      cancelButton = new javax.swing.JButton();
      clearButton = new javax.swing.JButton();
      
      setTitle("Edit Access List");
      addWindowListener(new java.awt.event.WindowListener() {
         public void windowClosing(java.awt.event.WindowEvent evt) {
            exitForm(evt);
         }
      });

      mainPanel.setLayout(new java.awt.GridBagLayout());
      
      mainPanel.setPreferredSize(new java.awt.Dimension(300, 200));
      userListLabel.setT ext("Add Users to the access list");
      
      gridBagConstraints = new java.awt.GridBagConstraints();
      
      userNameText.setEditable(true);
      userNameText.setColumns(10);
      userNameText.setRows(5);

      returnWindow.setVisible(true);
   }

   public static String returnUserList;
   public static FileIDgui returnWindow;
gridBagConstraints.gridwidth = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
mainPanel.add(userListLabel, gridBagConstraints);

userNameText.setColumns(10);
userNameText.setSize(new java.awt.Dimension(75, 160));
userNameText.setRows(4);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 4;
gridBagConstraints.gridheight = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
mainPanel.add(userNameText, gridBagConstraints);

okButton.setText("Ok");
okButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        okButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 5;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
mainPanel.add(okButton, gridBagConstraints);

cancelButton.setText("Cancel");
cancelButton.setMaximumSize(new java.awt.Dimension(52, 26));
cancelButton.setSize(new java.awt.Dimension(52, 26));
cancelButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        cancelButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 5;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
mainPanel.add(cancelButton, gridBagConstraints);

clearButton.setText("Clear");
clearButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        clearButtonActionPerformed(evt);
    }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 5;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
mainPanel.add(clearButton, gridBagConstraints);

getContentPane().add(mainPanel, java.awt.BorderLayout.CENTER);

pack();
} // end initComponents

private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
    userNameText.setText("");
}

private void cancelButtonActionPerformed(java.awt.event.ActionEvent evt) {
    returnWindow.messageText.setText("edit access list canceled");
    this.dispose();
}

private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String temp = new String(userNameText.getText());
    returnWindow.messageText.setText("Access list edited");
    this.dispose();
}

private void exitForm(java.awt.event.WindowEvent evt) {
    this.dispose();
}

// Variables declaration
private javax.swing.JButton okButton;
private javax.swing.JLabel userListLabel;
private javax.swing.JButton clearButton;
private javax.swing.JButton cancelButton;
private javax.swing.JPanel mainPanel;
private javax.swing.JTextArea userNameText;
} //end UserListEditor
Appendix H. FAMcrypto.java

```
/*
 * FAMcrypto.java
 */
package fileAccessManager;
import java.math.*;
import java.io.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;
import java.security.Key.*;
import java.security.spec.*;
public class FAMcrypto {
    public static FileIDgui mainWindow; //used to print error messages for user
    /**Creates a new instance of FAMcrypto */
    public FAMcrypto(FileIDgui parent) {
        mainWindow = parent;
    }

    public static int encrypt(String filename, BigInteger Blkey, String fileID) {
        int succeeded = 0;
        Cipher myCipher;
        File inputFile = new File(filename);
        File outputFile = new File(fileID + "enc");
        try {
            DESKeySpec ks = new DESKeySpec(Blkey.toByteArray());
            SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
            SecretKey myKey = skf.generateSecret(ks);
            myCipher = Cipher.getInstance("DES/CFB8/NoPadding");
            myCipher.init(Cipher.ENCRYPT_MODE, myKey);
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileID
            + "IV"));
            oos.writeChars(filename + " ");
            oos.writeChars(Blkey.toString() + " ");
            CipherOutputStream cos = new CipherOutputStream(new FileOutputStream(outputFile), myCipher);
            PrintWriter pw = new PrintWriter(new OutputStreamWriter(cos));
            FileInputStream tis = new FileInputStream(inputFile);
            BufferedReader hr = new BufferedReader(new InputStreamReader(tis));
            int length = (int)inputFile.length();
            char inputBuf[] = new char[(int)length];
            int bytesRead = br.read(inputBuf);
            int done = bytesRead;
            if (length == (long)bytesRead) {
```
pw.write(inputBuf);
} else while (done < length) {
    bytesRead = br.read(inputBuf);
    pw.write(inputBuf, done, done + bytesRead);
    done += (long)bytesRead;
}//file all read
pw.close();

oos.writeObject(myCipher.getIV());
oos.close();
} catch (NoSuchAlgorithmException alg) {
    mainWindow.messageText.setText("Algorithm error " + alg.getMessage());
succeeded = -1;
} catch (NoSuchPaddingException pad) {
    mainWindow.messageText.setText("Padding error " + pad.getMessage());
succeeded = -1;
} catch (IOException ioe) {
    mainWindow.messageText.setText("IOException " + ioe.getMessage());
} catch (Exception e) {
    mainWindow.messageText.setText("Error encrypting " + e.getMessage());
succeeded = -1;
}
return succeeded;
}//end encrypt

public static int decrypt(String filename, BigInteger key, String fileID) {
    int succeeded = 0;
    Cipher myCipher;
    File inputFile = new File(filename);
    File IVfile = new File(fileID + "IV");
    StringBuffer strBuf = new StringBuffer();
    try {
        ObjectInputStream ois = new ObjectInputStream(
                new FileInputStream(IVfile));
        char myChar = ois.readChar();
        while (myChar != ' ') {
            strBuf.append(myChar);
            myChar = ois.readChar();
        }
        String originalFilename = new String(strBuf.toString());
        File outputFile = new File(originalFilename);
        strBuf = new StringBuffer();
        myChar = ois.readChar();
        while (myChar != ' ') {
            strBuf.append(myChar);
            myChar = ois.readChar();
        }
        key = new BigInteger(new String(strBuf.toString()));
        //get IV
        IvParameterSpec ivSpec = new IvParameterSpec((byte[]) ois.readObject());
        DESKeySpec ks = new DESKeySpec(key.toByteArray());
        //init cipher
        try {
            if (ivSpec instanceof IvmParameterSpec) { // GCM
                GCMParameterSpec gcm = new GCMParameterSpec(128, ivSpec.getIV());
                myCipher = Cipher.getInstance("AES/GCM/NoPadding", "SunJCE"));
            } else {// get cipher
                myCipher = Cipher.getInstance("DES/CBC/PKCS5Padding", "SunJCE");
                myCipher.init(myCipher.createCipherSpec(ks));
            }
            myCipher.init(Cipher.DECRYPT_MODE, ks, ivSpec);
            byte[] len = myCipherSpi.getIV();
            for (int i = 0; i < len.length; i++)
                IVbytes[i] = len[i];
            myCipher.update(inputBuf, 0, IVbytes.length, IVbuffer);
            for (int i = 0; i < IVbytes.length; i++)
                IVbuffer[i] = IVbytes[i] == 0 ? IVbytes[i] : IVbytes[i] + len[i];
        } catch (Exception e) {
            mainWindow.printTextarea.setText("Error encrypting " + e.getMessage());
        }
    }
    String outputString = strBuf.toString();
    outputFile = new File(outputString);
    strBuf = new StringBuffer();
    myChar = ois.readChar();
    while (myChar < ' ') {
        strBuf.append(myChar);
        myChar = ois.readChar();
    }

    for (int i = 0; i < IVbytes.length; i++)
        IVbytes[i] = IVbytes[i] % 256;
    for (int i = 0; i < IVbytes.length; i++)
        IVbuffer[i] = IVbytes[i] == 0 ? IVbytes[i] : IVbytes[i] + len[i];

    outputString = strBuf.toString();
    System.out.println(outputString);
    try {
        outputFile = new File(outputString);
        strBuf = new StringBuffer();
        myChar = ois.readChar();
        while (myChar != ' ') {
            strBuf.append(myChar);
            myChar = ois.readChar();
        }
        key = new BigInteger(new String(strBuf.toString()));
        //get IV
        IvParameterSpec ivSpec = new IvParameterSpec((byte[]) ois.readObject());
        DESKeySpec ks = new DESKeySpec(key.toByteArray());
        //init cipher
        try {
            if (ivSpec instanceof IvmParameterSpec) // GCM
                GCMParameterSpec gcm = new GCMParameterSpec(128, ivSpec.getIV());
            else // get cipher
                myCipher = Cipher.getInstance("DES/CBC/PKCS5Padding", "SunJCE");
        } catch (Exception e) {
            mainWindow.printTextarea.setText("Error encrypting " + e.getMessage());
        }

    } catch (Exception e) {
        mainWindow.printTextarea.setText("Error encrypting " + e.getMessage());
    }
} catch (Exception e) {
    mainWindow.printTextarea.setText("Error encrypting " + e.getMessage());
}
SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
SecretKey myKey = skf.generateSecret(ks);

myCipher = Cipher.getInstance("DES/CFB8/NoPadding");
myCipher.init(Cipher.DECRYPT_MODE, myKey, ivSpec);

CipherOutputStream cos = new CipherOutputStream(
    new FileOutputStream(outputFile), myCipher);
PrintWriter pw = new PrintWriter(new OutputStreamWriter(cos));
FileInputStream fis = new FileInputStream(inputFile);
BufferedReader br = new BufferedReader(new InputStreamReader(fis));
int length = (int)inputFile.length();
char inputBuf[] = new char[(int)length];
int bytesRead = br.read(inputBuf);
int done = bytesRead;
if (length == (long)bytesRead) {
    pw.write(inputBuf);
} else while (done < length) {
    bytesRead = br.read(inputBuf);
    pw.write(inputBuf, done, done + bytesRead);
    done += (long)bytesRead;
} //file all read
pw.close();
} catch (NoSuchAlgorithmException alg) {
    mainWindow.messageText.setText("Algorithm error " + alg.getMessage());
    succeeded = -1;
} catch (NoSuchPaddingException pad) {
    mainWindow.messageText.setText("Padding error " + pad.getMessage());
    succeeded = -1;
} catch (IOException ioe) {
    mainWindow.messageText.setText("IOException " + ioe.getMessage());
} catch (Exception e) {
    mainWindow.messageText.setText("Error encrypting " + e.getMessage());
    succeeded = -1;
}
return succeeded;
}//end decrypt
}//end FAMCrypto
References


