Final Software Design Specification

Version 1.0

Version History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date | 11/4/2009 | Authors | Anthony Sinatra  Dmitry Sharlot  Joseph Everett  Ricardo Viera | Version | 1.0 |
|  | 11/09/2009 |  | Updated with implementation details of Encryption along with encryption diagrams | | |
|  | 11/22/2009 |  | Updated Login, Encryption, Decryption, and Help Screenshots | | |
|  | 11/22/2009 |  | Added Java Virtual Machine constraint | | |
|  | 11/22/2009 |  | Added Component Level Representation diagram | | |

1. **Introduction**
   1. **Goals and Objectives**

The purpose of FogWire system is to ensure the protected transmission of electronic mail between a sender and the intended recipient. The project will encompass all the principals of the software development lifecycle from the requirements engineering to design, implementation, and testing of a complete functional software system. The system will use encryption and decryption technologies which deem the electronic mails data unusable by unauthorized parties. The software will allow authorized parties to securely transmit data between each other.

* 1. **System Statement of Scope**

FogWire is a desktop application which transforms electronic mail data into indecipherable text. A user with any knowledge of electronic mail applications can use the FogWire system. Upon first execution of the application, the user will enter profile information which will be stored. Only authorized users will have access to the system. The user interface will allow users to easily navigate through the process of encrypting or decrypting their electronic mail data.

Encryption data will be entered using copy/paste, free-text or text file upload functionalities and a Graphical User Interface (GUI). Decryption data will be entered using a file upload functionality of the encrypted text file. Output will be stored in a file to a local drive

Information stored about the user includes:

* First Name
* Last Name
* Email Address
* Public Key
* Private Key
  1. **System Context**

FogWire is presented as an application which will be used by any general computer user to securely transmit data to any other general computer user via electronic mail. It is a user friendly application which will not require any training in order for the user to successfully accomplish her/her desired task.

**Potential Users**

Potential users include pre-teens, teenagers, young adults, adults, and even seniors who have at least a novice level of understanding of electronic mail applications and associated tasks.

**Teenagers:** The system will allow this age group to send each other data without risk of outside involvement (i.e. from parents or other teenagers) and can even be used on public networking sites using cut and paste from the encryption area.

**Young Adults, Adults, and Senior Citizens:** The system will allow this set of users to send data to intended recipients without the risk of outside involvement (i.e. from identity thieves) and can even be used to send any type of confidential information (i.e. credit cards numbers, account information, medical history).

* 1. **Major Constraints**

The user will be required to have a connection to internet. Also, he/she must have a valid e-mail address.

A Java Virtual Machine (JVM) version 6.0 is also required for application execution.

The working prototype of the system will be available December 16, 2009 at the end of the semester.

1. **Data Design**

**2.1 Internal Software Data Structure**

**2.1.1 Major Data Objects**

The following data modules will be managed by the system:

* User Creation Module
  + This will create a new user and his/her public key and private key. It also authenticates the user as a valid FogWire application client.
* Lock Data Module
  + Accepts copy/pasted data or a text file and transforms this into encrypted data.
  + Inserts public key into the encrypted file in order to allow for decryption.
  + Sends the encrypted file using FogWire account setup on the Google Mail client.
* Unlock Data Module
  + Accepts an encrypted file
  + Transforms the encrypted data back to its decipherable original state which the sender intended for only the authorized recipient to view. This is accomplished by providing the public key.
* Help Module
  + Displays user quick tips of the Lock Data process and workflow
  + Displays user quick tips of the Unlock Data process and workflow

**2.1.2. Relationships**

**Senders**

**Fogwire**

**Application**

Via Google Mail Client

**Authorized Recipients**

**2.2 Database Description**

**Database Relationship Diagram**

Following is the ERD for FogWire System and its description.

ERD for User Database

**Users**

**User Database**

The user database will hold information about each user in order to verify that he/she is an authorized FogWire user, when encryption and decryption is performed.

**Users Table**

The table will store information about each user such as first name, last name, e-mail address and public key.

**User Database Interface Description**

The interface will only be available upon the first application execution. It will allow the user to enter first name, last name, and e-mail address. It will then allow the user to access the Lock Data/Unlock Data/Help area.

**Interfaces** **and access to FogWire Encryption/Decryption**

These interfaces will be available to users who have been authenticated within the user’s database. They will include lock data, unlock data and help screen.

**Interface Description**

**Lock Data Screen**

Data can be entered through copy/paste, free-text, or file upload in order to be encrypted. The user can then save it to a file or save and send to another authorized FogWire recipient.

**Unlock Data Screen**

Data can be entered through file upload in order to be decrypted and available to the user.

**Help Screen**

This area will display the proper steps to Lock and Unlock documents.

- Anthony please expand on the instructions here

**2.3 Database Structure**

|  |  |  |
| --- | --- | --- |
| Entity **Users** | | |
| **Attribute** | **Type** | **Description** |
| e\_mail | VARCHAR(10) | Primary key, the user’s e-mail address |
| f\_name | VARCHAR(20) | User’s first name |
| l\_name | VARCHAR(30) | User’s last name |
| pub\_key | VARCHAR(32) | User’s public key which is used during encryption |

1. **Component Level Design**

**3.1 Program Structure**

Java will be used to build the FogWire user interface. Java technology will also be used to perform the encryption/decryption actions and connect to the database.

**3.2 Description User Profile Setup**

**3.2.1 Processing Narrative for User Profile Setup**

When the application is first initialized, the user must enter first name, last name and e-mail address. Upon future login, the user details will be authenticated against the local files stored in the users FogWire folder. If the user exists, he/she will proceed to the lock data/unlock data areas. If the user does not exist, he/she will have to setup a user profile before proceeding. When the profile is saved, the users’ public key will automatically be generated and stored to the database without informing the user. The users private key will be also be generated and stored locally without informing the user.

**3.2.2 User Profile Setup Interface Description**

Input Interface – First Name, Last Name, E-mail Address

Output Interface – ­ If the profile was set up the user will be automatically directed to the main lock data/unlock data screen . Otherwise the user will receive a message with guidance on how to properly register.

**3.2.3 User Profile Processing Detail**

**3.2.3.1 Algorithmic model (PDL)**

Procedure:

Lookup if user profile exists

If profile exists and is valid

Return user to Lock Data/Unlock Data screen

Else return user to User Profile Setup Screen

**3.2.3.2 Restrictions/Limitations**

The method of user authentication does not prevent physical machine access.

**3.2.3.3 Local Data Structures**

The local data structures used in the FogWire System are in order to store the users’ information and private key. This is linked to the users’ profile information stored in the database.

**3.2.3.4 Performance Issues**

None

**3.2.3.5 Design Constraints**

None

**3.3 Description for Encryption**

**3.3.1 Processing Narrative for Encryption**

The locking data area is responsible for receiving all data which is to be encrypted. It is

responsible for receiving this data via copy/paste, free-text, or file upload. It is also

responsible for accepting the recipients e-mail address which should receive the encrypted file.

Does the below still hold true Ricardo? This needs some revision.

**3.3.2 Lock Data Interface Description**

Input interface – text box for accepting data to lock, textbox to accept the recipient’s e-mail address, browse for file button to find a file to lock.

Output interface –A flag indicating the process was successful.

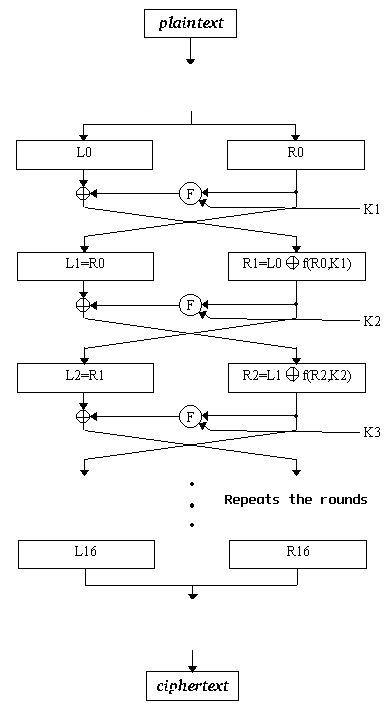
**3.3.3 Lock Data Processing Detail**

The Encryption Process – A file is read at the receiver’s FogWire application. As the file is read, an encryption method is acted on the characters of the file. The character that is altered is then written on another file that is Stored in Fogwire/Locked\_Docs folder, etc.

The encryption method is a derivative of the DES (Data Encryption standard) process. FogWire’s version will take in a 32 bit character and split the binary bit representation of this character into 2 separate binary integers. One will be the (Li) left integer of our original character and the other will be the (Ri) right integer. A copy of Ri (copyRi) will traverse as a parameter into the (innerF()) inner function and be encrypted with the recipient’s public key. The result from the innerF() function will be returned to the main Encryption process, being described, and be ‘Xored’ with our left integer (Li). ‘Xored’ is a term used to describe a mathematical function called exclusive or. It is when a bit in

binary is compared to another bit in binary. If the bits are different then the Xor function produces a 1 bit; otherwise if the bits are similar then the 0 bit is produced. The result of Li Xor innerF(Ri, key) becomes our new Li() and Ri, remains our initial Ri from the original binary representation of the character entered into this encryption process. Our new Li and our original Ri are swapped. That is, that our new Li is labeled Ri and our original value Ri is now labeled Li. The process is repeated 15 more times and the value returned is our encrypted character. As the characters are encrypted they are placed on a file to be sent to our recipient.

**Encryption Process**



Decryption Titles here?

**3.4 Description for Decryption??**

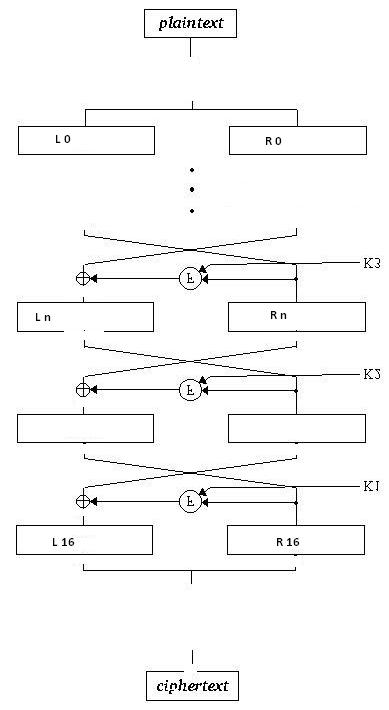
**3.4.1 Processing Narrative for Decryption**

**3.4.2 Unlock Data Interface Description**

**3.4.3 Unlock Data Processing Detail**

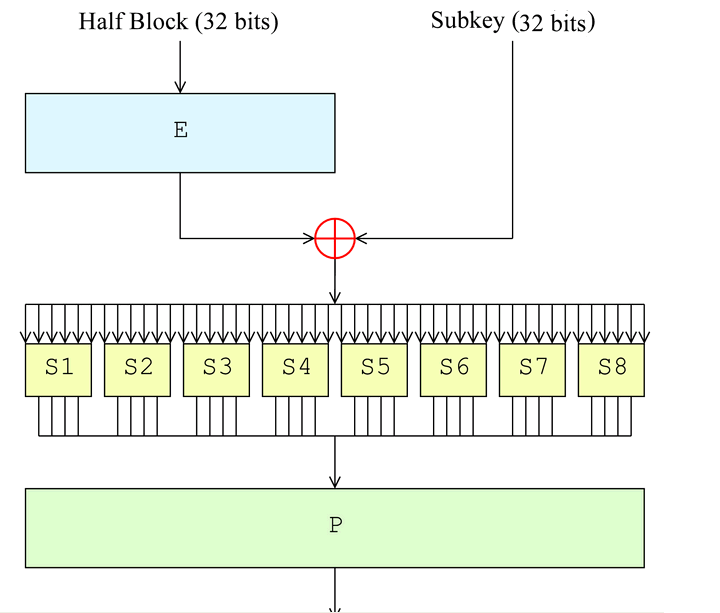
**This needs to be filled in.**

**Decryption Process**



Revision?

The innerF() inner function is a process in which the Ri right integer is Xored with the Key generated from the recipients public key. After our Ri is Xored with the key, that value is then pushed through eight, 4-bit substitution box components. The substitution box will take 4 bits of the Ri key value and substitute those 4-bits with a predetermined value set by FogWire’s programmers. The 8 substitution boxes will act on the 32 bit binary integer representation of the character. After the Ri key value is substituted the new substituted result is then permuted before being returned to the original encryption function. A permutation is taking the current sbox(Ri⊕key) value and moving the same bits around into different position within the binary representation of the integer that is the value of the character. This value pBox(sbox(Xor(Ri,key))) is then return to our encryption process and will be Xoared with Li, our left integer value from the original binary integer representation of our initial character from the start of our encryption process of the initial file read.



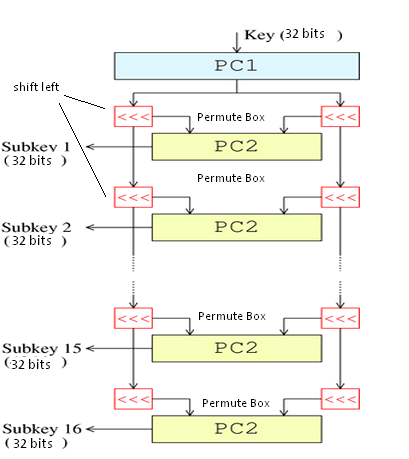
**3.5 Key Generation??**

Key generator revision needed. + proper titles.

Key generation will be a process in which RSA encryption will initially, one time, generate a 32 bit key from a Random choice value. The key generated by the RSA encryption process which will take a value p and another q that are prime numbers and multiply them. That multiplication will result with a much larger value n. Then we will take the values (p-1)(q-1) and create a value phiof(n). Then we will select a value e such that e is between the values 1 and phiof(n). We will send out a value encrypted with Public\_key(e,n) as our public key. We will keep a value d as our private key. The value d is generated by taking the inverse of e in modulo phiof(n).

**NOTE:** In DES, Data Encryption Standard, the 16 rounds will require a key for each round and that key will be generated from the one initial public key. The additional key rounds will be created by a key generator. Therefore the initial key will enter the roundkey generator and produce 16 keys for each round. The 32 bits will become 2 integers Li and Ri like before. The function shift left will be acted on each integer. The 2 integers are combined and that 32bit shifted combination is permuted. That pbox(concat(shiftleft(Li),shiftleft(Ri))) value will produce the round key that is used in the innerF() of our initial Des encryption.

**Diagram still ok?**



1. **User Interface Design**

**4.1 Description of User Interface**

The following sections explain the basic screens which are necessary for collecting data

for locking data and unlocking data.

**4.1.1 Screen Images**

**User Profile Setup Screen**

The first screen that the user will see upon the first initialization of the FogWire application. The user profile setup screen allows the user to enter first name, last name, and e-mail address. It has inputs for all three fields (first name, last name, e-mail address) and a button to submit this information

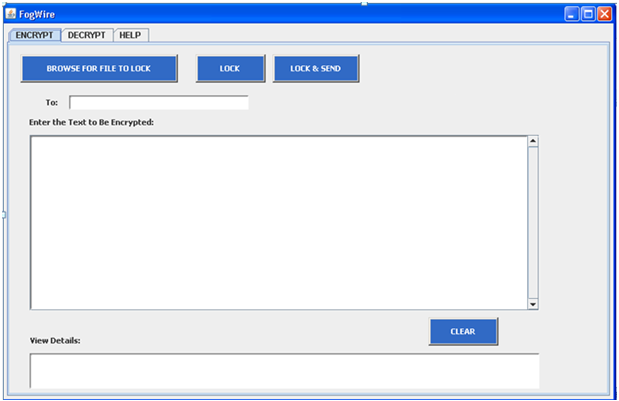
**ALL SCREENSHOTS NEED TO BE UPDATED!!!!!**

****

**Loc k Data Screen**

After the user has completed the profile information or upon subsequent application execution, he/she will be taken to the Lock Data screen. There will be a text box where the user can submit data to be encrypted by selecting LOCK ONLY button. The user can select a file using the ‘Browse for File toLOCK’ functionality. Also, the user can enter e-mail address of the intended recipient, if using the ‘LOCK & SEND functionality. The user can also clear the text box using ‘CLEAR’ button

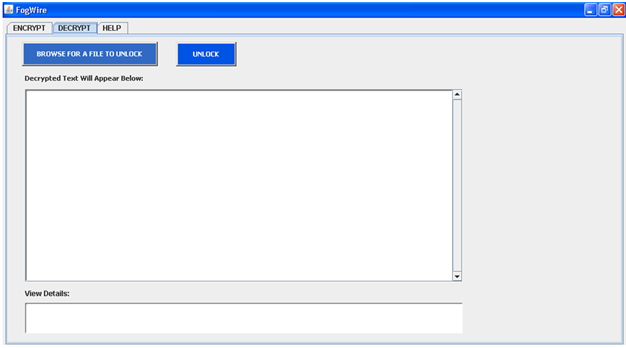
**ALL SCREENSHOTS NEED TO BE UPDATED!!!!!**



**Unlock Data Screen**

By navigating using tabs, the user will be taken to the Unlock Datascreen by selecting the ‘UNLOCK DATA’ tab. The user can select a file to decrypt from the FogWire default folder using ‘Browse for a File toUnlock. The user can then select the ‘Unlockbutton which will display the decrypted text on the screen and/or store the decrypted file in the FogWire default folder using the ' this just ends – are we missing part of a sentence??

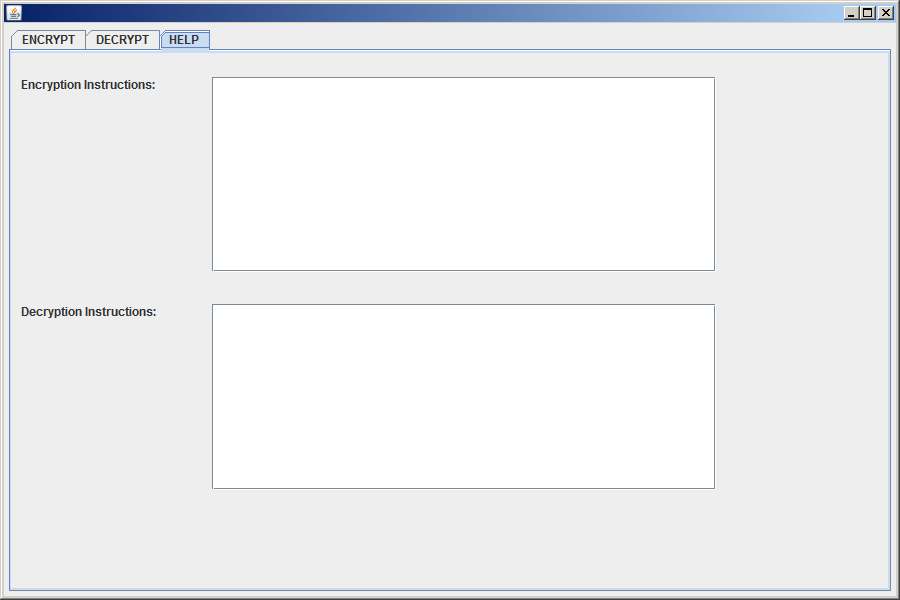
**ALL SCREENSHOTS NEED TO BE UPDATED!!!!!**



**Help**

This help area will display quick instructions to the user on how to perform any relevant actions in FogWire system.

**ALL SCREENSHOTS NEED TO BE UPDATED!!!!!**



**5.0 Restrictions, Limitations, Constraints**

The following items include the limitations and constraints that are part of the specification:

1. The application will not function properly unless the user is connected to the internet
2. The application must have a connection to the database in order for the system to verify and/or store the users’ information.
3. Only FogWire users will be able to Unlock data which has been sent from other FogWire users
4. FogWire e-mails will be sent using the [FogWire@gmail.com](mailto:FogWire@gmail.com) domain
5. The subject line of e-mails sent using FogWire will default to “You have a new message from FogWire!”
6. The user must have the Java Virtual Machine (JVM version 6.0) installed on his/her computer in order to run FogWire.

**6.0 Validation Criteria**

**6.1 Description for software behavior**

We will be using the following software testing methodologies in order to verify that the functionality meets the requirements and uncover any errors or flaws which exist.

1. Functional Testing
2. Regression Testing

Functional testing is used to verify that the requirements have been met. Functional testing is constructed using realistic test scenarios which have been prepared after careful business analysis. Each test case will be created from the original user requirements and be executed on the software system. A test case will be considered failed if the requirement has not been met or if the system does not perform according to the test scenario.

Regression Testing will be used to uncover any remaining errors after functional testing is complete. Regression testing follows a set of user actions or test scenarios which may be likely to occur. If an error is uncovered, the business analyst will decide whether it is a realistic test scenario which is likely to occur following a realistic set of user actions. If the scenario is likely, the error will be fixed. If it is unlikely, the error will be added to the list of known issues or will be scheduled into a subsequent version.