

FogWire Test Plan

Specification

Version 1.0

Version History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date | 12/05/2009 | Authors | Anthony Sinatra  Dmitry Sharlot  Joseph Everett  Ricardo Viera | Version | 1.0 |
|  | 12/06/2009 |  | Updated with Interface testing 2.2.1  Updated with Component testing 2.2.2 | | |
|  | 12/09/2009 |  | Updated Component testing 2.2.2 | | |

**FogWire Personal Encryption Software- Test Plan Specification**

**Table of Contents**

**1.0 Introduction**

**1.1 Goals and objectives**

**1.2 Statement of scope**

**1.3 Software context**

**1.4 Major Constraints**

**2.0 Test Plan**

**2.1 Overview**

**2.2 Software Testing**

**2.2.1 Interface Testing**

**2.2.2 Components Testing**

**3.0 Test Procedure**

**3.1 Testing Procedures**

**3.1.1 Functional Testing**

**3.1.2 Validation Testing**

**3.1.3 Regression Testing**

**3.1.4 High-order Testing**

**3.2 Testing Resources and Staffing**

**3.3 Test Record Keeping and Test Log**

***1. Introduction***

The FogWire Personal Encryption software system provides the users with a simple way of encrypting data and sending this data to the intended recipients. It also allows the recipients to decrypt this data and store it on their personal computer. The FogWire system will consist of an encryption screen and a decryption screen. The encryption screen allows the user to lock his/her data and either store it on his/her computer or store and send to another FogWire user. The decryption screen allows the user to unlock data which has been previously encrypted by another FogWire user.

All FogWire authorized users have the ability to encrypt and decrypt text. However, in order for users to decrypt text which has been sent by another FogWire user, both the sender and the recipient must share the same Fog Code. The Fog Code can be changed before encryption or decryption and can be used to create security amongst a particular group or groups of users.

* 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 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.

***1.2 Statement of Scope***

FogWire is a desktop application which transforms electronic mail text 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 files. Output can be stored in a file to a local drive.

***1.3 Software 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 his/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).

***1.4 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.

**2.0 Test Plan**

***2.1 Overview***

A dependable test plan will be used to uncover any errors which may exist before the actual software deployment and final user acceptance. Software errors which appear later in the software development process are more costly to the project as a whole. These errors must be discovered and resolved earlier in the software development process in order to ensure that the users’ experience is not affected by poor quality, missing functionality, or low usability.

***2.2 Software Testing***

The software is divided between the interface, encryption/decryption, database, and a mail client. The encryption area will take any given text (through file upload or text box entry) and perform a series of functions on the particular text. These functions will deem the text unusable and indecipherable by any unauthorized parties. The text will only be made usable and decipherable when it has been passed through the decryption area using the same Fog Code used during the encryption process.

The following areas need to be tested: the interface which is displayed to the user, the encryption/decryption components, the database components, and the mail client components.

**2.2.1 Interface Testing**

***Authentication***

Allows the user the ability to register his/her FogWire system if he/she has not done so already

Example Authentication test cases include:

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Register | Enter data in the text fields and click ‘Register’ | The application opens. The user is presented with the ‘Lock Data’ area. |
| Missing Registration Data | Enter data with any field (First Name, Last Name, E-mail Address, and/or Fog Code) blank | The application does not open. The user receives prompt ‘All fields are Required to Register!’ <OK> |
| Invalid E-mail Address (Authentication) | Enter an e-mail address that is not RFC 2822 compliant | The application does not open. The user receives prompt ‘Email Address is invalid!’ <OK> |

***Lock Data***

Allows the user the ability to lock text and save it or lock/save and send it to another

FogWire user

Example Lock Data Test Cases Include

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Browse for File to Lock | User selects ‘Browse for File to Lock’ button | User receives a window with the folders from his/her local computer. |
| Enter text and Lock | User enters text in the text area and clicks Lock. Browse button is disabled. | User receives a prompt ’File Successfully locked!’ |
| Browse for non-text file to Lock | User attempts to select a file that does not end in .txt | User receives a prompt ‘Invalid File type. Please select a Text file (\*.txt)’ |
| Select File to Lock | User selects a .txt file and selects ‘Open’. | The data displays in the ‘Enter the text to be locked’ textbox |
| Lock and Send No E-mail Address/Invalid E-mail Address | User selects ‘Lock and Send’ button with no e-mail address | User receives a prompt ‘Email Address is invalid’ <OK> |
| Lock and Send E-mail Address is not a valid FogWire user | User selects ‘Lock and Send’ button with an e-mail address entered which is not a valid FogWire user. | User receives a prompt ‘Email Address is invalid’ <OK> |
| Lock Only with no text | User selects ‘Lock Only’ without entering data | User receives a prompt ‘No Data to Lock’ <OK> |
| New Fog Code | User selects ‘New Fog Code’ button | User receives a window to ‘Enter New Fog Code’ and select <Submit> |
| Lock text using one Fog code, attempt to unlock text using another fog code | User enters text and clicks Lock. User clicks New Fog code Button and enters a new fog code. User browses for the locked file and clicks unlock | Data does not unlock. Data is unreadable |
| New Fog Code number not in the range 10,000 – 59,999 | User attempts to enter a fog code less than 10,000 or greater than 59,999 | User receives a prompt ‘Fogcode is invalid! Please enter a fogcode between 10,000 – 59999’ |
| Clear (Text Area) | User selects ‘Clear’ button | Any data displayed in the ‘Enter Data to Lock’ text box is cleared |
| View Details | User performs a Lock or Lock & Send operation | View Details area updates with information about the procedures |

**Unlock Data**

Allows user the ability to unlock text and display it

Example Unlock Data Test Cases Include

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Browse for File to Unlock | User selects Browse for File to Unlock | User receives a window with the list of folders from his/her local computer. The default folder is ‘Locked\_Docs’ |
| Select a File to Unlock | User selects a .txt file and selects ‘Open’ | The user is returned to the Unlock Data tab, the encrypted text is displayed and the ‘Unlock’ button now appears under the Unlock Data tab |
| Unlock Button | User selects ‘Unlock’ button | The unlocked data displays in the ‘Unlocked Text will appear below’ text box. The Unlock button is disabled until the user browses for another document to unlock. |
| Browse for file non-text file to Unlock | User attempts to select a file that does not end in .txt | User receives a prompt ‘Invalid File type. Please select a Text file (\*.txt)’ |
| View details | User browses for a file and unlocks the data | View Details area updates with information about the procedure |

**Help**

Displays a list of instructions on how to Lock Data and Unlock Data

Example Help Test Case Include

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Help | User selects the ‘Help’ tab | Two areas display ‘Data Locking Instructions’ and ‘Data Unlocking Instructions’. Both areas display text which is not editable. |

**2.2.2. Component Testing**

**CheckFolder**

The CheckFolder component performs a function which checks to see if the Foggy.txt file exists in C:\FogWire directory. This will decide if the user previously registered with FogWire. Then, it will authenticate the user and proceed to the application. If the user has not previously registered, FogWire will allow the user to register by providing the registration screen.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Check Foggy File | Open FogWire application and submit the required data | Open the Foggy.txt file and verify the existence of the data entered |
| Application Open | Open the application (After having previously submitted registration data) | Verify that the application opens with the Lock Data tab displayed. |

**Add User**

The Add User component retrieves the user information from the registration screen and stores it in the local Foggy.txt file as well as the FogWire database. This component also generates RSA Keys which are placed in the local Foggy.txt file as well as on the FogWire database for the Authenticate User Component to use.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Check Database | Open FogWire application and submit the required data to register | Connect to the database, load the drivers, and run a query using standard SQL select statement. Verify the data exists correctly in the database. |

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Check RSA Keys | Open FogWire application and submit the required data to register | Open the Foggy.txt file and verify the existence of the generated key data.  Connect to the database, load the drivers, and run a query using standard SQL select statement. Grab the generated public key data for the user off of the database as well as the private key data stored locally in the Foggy.txt file and run our algorithm to verify the correct pair of keys. |

**Authenticate User**

The Authenticate User component retrieves part of the user information from the Foggy.txt file stored locally as well as retrieves part of the user information stored at the FogWire database and then runs a number of authentication checks to verify the user as well as the local application.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify User | Open FogWire application | Connect to the database, load the drivers, and run a query using standard SQL select statement. Verify the data exists correctly in the database and matches the data stored locally on the computer. |

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify Application | Open FogWire application | Connect to the database, load the drivers, and run a query using standard SQL select statement. Grab the public key data for the user off of the database as well as the private key data stored locally in the Foggy.txt file and run our algorithm to verify the correct pair of keys. |

**Send Mail**

The Send Mail component retrieves locked.txt file from the C:FogWire\Locked\_Docs folder and sends it to the identified recipient via Gmail client.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Send Mail | Open FogWire application. At the Lock Data tab select a file to be locked and press Lock and Send button. | Connect to the database, load the drivers, and run a query using standard SQL select statement. Verify that the recipient is a registered FogWire user. Send the locket.txt file as an attachment to the user. Receive “Successfully Sent Mail” pop up confirmation. |

**Encrypt**

The encryption DES process used to transform regular text into something illegible but recoverable.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify Encrypt | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**DesEncrypt Rounds**

DesEncrypt Rounds is the process of setting a fix amount of DES encryption process repetitions.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify DesEncrypt | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**DesEncrypt**

DesEncrypt is the Encryption process of reading a character and performing the properties of diffusion and confusion by mutating bit values through xoring, permutation, substitution, and repetition.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify DesEncrypt | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**InnerFunction**

InnerFunction is a function in which a character value is encrypted by using the methods of xor, substitution, and permutation.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify InnerFunction | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**EComponents**

EComponents is the container class used to house the methods of the components needed in the DES process.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify EComponents | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**PBOX**

PBOX is a permutation chart that moves swaps the position of a bit with a known other position of a bit.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify PBOX | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**SBOX**

SBOX is a substitution chart used to substitute a known bit position with a chosen bit into that particular position.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify SBOX | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**XOR**

XOR is the Boolean operation of comparing two bits and using a bit value to represent their comparison.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| VerifyXOR | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**Decrypt**

Decrypt Component reads in a character from a file and performs the DES encryption process

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify Decrypt | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**DesDecrypt Rounds**

DesDecrypt Rounds runs the DES decryption process a preset amount of time.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify DesDecrypt Rounds | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**DesDecrypt**

DesDecrypt is the process of using the bit value of the character representation and rearranging the bit value to create diffusion and confusion of the file to be read.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify DesDescrypt | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**InnerDFunction**

The InnerDFunction is a function of bit xor’ing, permutation, and bit substitution.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify InnerDFunction | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**EComponents**

The EComponents is the container class of the DES methods that perform decryption and encryption.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify EComponents | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**PBOX**

A permutation chart that moves swaps the position of a bit with a known other position of a bit.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify PBOX | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**SBOX**

SBOX is a substitution chart used to substitute a known bit position with a chosen bit into that particular position.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify SBOX | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**XOR**

XOR component is the Boolean operation of comparing two bits and using a bit value to represent their comparison.

|  |  |  |
| --- | --- | --- |
| **Test Case Name** | **Test Case Actions** | **Test Case Expected Results** |
| Verify XOR | Enter value into the component code. Take output and run through reverse component | Verify output received from reverse process matches value entered into original process |

**3.0 Test Procedure**

This section will describe more detailed information about the testing processes which have been implemented for the FogWire system.

***3.1 Testing Procedures***

***3.1.1 Functional Testing***

Functional testing is performing in order to ensure that the requirements have been satisfied. Each piece of functionality is tested and traced back to one of the original requirements defined for the FogWire system. Functional testing will test each individual area which was introduced within the software to satisfy any of the specified requirements. The software has been divided into separate areas of functionality, which are Registration, Unlock Data, Lock Data, and Help.

***3.1.2 Validation Testing***

Validation testing is also known as black box testing. It is used to validate the software using the appropriate business and usability rules; however, the internal components cannot be seen. This testing will validate that the specified business requirements and functionality has been satisfied. Test data selection and the interpretation of the test results are based solely on the functional properties of the software and not the internal workings of it.

***3.1.3 Regression Testing***

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.

***3.1.4 High- order testing***

These tests are performed with a combination of several different test methods:

**Security Testing**

Security testing will be performed to ensure that unauthorized users cannot decipher encrypted data. Only FogWire users which have registered with FogWire and have been designated by an intended recipient will have the ability to decrypt the data.

1. Registration

This forces users to provide the necessary information before using FogWire.

1. Fog Code

Ensures that only intended FogWire recipients can decrypt the encrypted text. Both users must have the correct FogCode in order to decrypt.

**Performance Testing**

The system should meet specific performance and efficiency requirements which will be tested. All operations performed by FogWire should be executed in a reasonable amount of time.

Performance testing will be performed using large amounts of data to ensure that the user experience is not affected by poor response times. Performance bounds will be considered for

1. FogWire Registration
2. Application execution
3. Encryption of file
4. Decryption of a file

**Alpha Testing**

Before the software is released, it will be tested within the software development group. All bugs found will be raised with the business analyst to determine the critically of any items found. Development will also ensure the software structure is sound. Business analyst will test to ensure that the business requirements have been satisfied and functionality is correct.

**Beta Testing**

Before the software is released, it will be tested by an educated user group. The purpose of Beta testing is to validate: User requirements/feature, End to End functionality, Work center methods and procedure, O A & M (operation, administration, maintenance) procedure, Data quality, System security, Measures of success, Recovery procedure. Beta testing will be done in the last week before the software if presented.

***3.2 Testing Resources and Staffing***

**Resources**

This section should include a listing of all the resources (e.g., test environment,

application executable, test tools, bug reporting utility, etc) necessary to perform the

testing.

2 PCs with open internet access will be used to test sending secure data from one user to another

The Java Virtual Machine version 6.0

Excel will be used for the test script execution and the process to report existing bugs

**Staffing**

Software Testing Coordinator – Anthony M Sinatra

Encryption/Decryption Component Testing Coordinator – Ricardo Viera

GUI Component Testing Coordinator – Joseph Everett

Database/Mail Client Component Testing Coordinator – Dmitry Sharlot

***3.3 Test Recording and Test Log***

An Excel Spreadsheet will be used to record all the results of the test cases performed. Any bugs or issues which are found will be tracked. Some will be scheduled for subsequent releases while other higher priority/business critical items will be fixed in this release. Each bug will be assigned an individual number for tracking purposes in the bug log.