Test Report for JScrypt

Jean Lucas Ferreira, Ocean Cheung, Harit Patel

December 8, 2015

Contents

1 Introduction **3**
1.1 Overview 3

2 Functional Requirements Testing **3**
2.1 Overview 3
2.2 Test Results 3

3 Non-Functional Requirements Testing **9**
3.1 Usability Requirements: 9
3.2 Performance / Speed Requirements: 9
3.3 Robustness: 10
3.4 Operational and Environment: 10

4 Testing Summary **10**
4.1 Changes Summary 10
4.2 Changes Implemented 11
4.3 Traceability Summary 11
4.4 Code Coverage Summary 11

5 Revision History **12**

1 Introduction

1.1 Overview

This test report is designed to summarize the testing JScrypt undertook, through a series of automated testing, and manual testing (for the front-end of the project). Valid and abnormal inputs have been tested within appropriate test cases, and changes made to the program in response to the test results have been documented throughout the report.

Automated testing was crucial for testing this project, and was achievable through the use of Mocha.js and Chai.js, which testing frameworks for NodeJS applications. It allowed for a quick implementation of testing, and helped find small issued with the program, which would not have been caught without automated testing (refer to Testing Conclusion for changes in the project).

A web site has also been constructed for this project, to allow for an easier visualization of how the program works, and to see what are outputs given certain inputs. Testing of this section will be done through a manual testing approach, and a component testing, by which the clickability and response of buttons on the website will be visually inspected.

2 Functional Requirements Testing

2.1 Overview

Testing of the functional requirements was created through a series of automated unit testing, while following a mix of black-box and white-box testing approach. The black-box testing was used to verify certain functional requirements, and assert that correct valued were being returned on a given input. While white-box testing helped in finding issues with the code that was not taken in consideration during development, and was not specified in the functional requirements.

2.2 Test Results

Tests were split up into two sections, one for JScrypt.js which is responsible for handling user input.The second section is Eksblowfish.js which is responsible for the encryption algorithm.

Module Tested: JScrypt.js
These are the initialized global variables in this module:
Global Variables:
BCRYPT_VERSION = '2a'
DEFAULT_ROUNDS = 10
MIN_ROUNDS = 6
MAX_ROUNDS = 31
SALT_LENGTH_BYTE = 16
SALT_LENGTH_CHAR = 22
KEY_HASH_SIZE = 31
MIN_KEY_SIZE = 1

MAX_KEY_SIZE = 56

Test File: JSCrypt.test.js
Test Unit: generateRandomSalt tests

Table 1: generateRandomSalt tests

Test Case #	Initial State	Input	Expected Output	Result
1	-Global variables initialized -Local variables declared but not initialized	rounds = 10	A variable that is of type String	Pass
2	-Global variables initialized -Local variables declared but not initialized	rounds = 10	A random string of 22 characters	Pass

Test Unit: hashKey tests

Table 2: hashKey tests

Test Case #	**Initial State**	**Input**	**Expected Output**	**Result**
1	-Global variables initialized -Local variables declared but not initialized	key = ‘ ’ rounds = 8	null	Pass
2	-Global variables initialized -Local variables declared but not initialized	key = ‘superSecretKey’ rounds = 1	A variable that is of type String	Pass
3	-Global variables initialized -Local variables declared but not initialized	key = ‘superSecretKey’ rounds = 8	Variable that is a hashed string	Pass
4	-Global variables initialized -Local variables declared but not initialized	key = null rounds = 8	null	Pass

Test Unit: getComponents tests

Table 3: getComponents tests

Test Case #	Initial State	Input	Expected Output	Result
1	-Global variables initialized -Local variables declared but not initialized	key = ‘’	Empty Array - []	Pass
2	-Global variables initialized -Local variables declared but not initialized	key = ‘$2b$10$IpocdZqL9TA8ZW2EWvpBJAa5w1QjNqmxAAAAAAGqoVPw==’	Empty Array - []	Pass
3	-Global variables initialized -Local variables declared but not initialized	key1 = ‘$2a$34$IpocdZqL9TA8ZW2EWvpBJAa5w1QjNqmxAAAAAAGqoVPw==’ key2 = ‘$2a$04$IpocdZqL9TA8ZW2EWvpBJAa5w1QjNqmxAAAAAAGqoVPw==’	Output 1: Empty Array - [] Output2: Empty Array - []	Pass
4	-Global variables initialized -Local variables declared but not initialized	key = ‘$2a$10$IpocdZqL9TA8ZW2EWvpBJAa5w1QjNqmxAAAAAAGqoVPw%%==’	Empty Array - []	Pass
5	-Global variables initialized -Local variables declared but not initialized	key = ‘$2a$10$IpocdZqL9TA8ZW2EWvpBJAa5w1QjNqmxAAAAAAGqoVPw==’	Array: [‘2a’,10, ‘IpocdZqL9TA8ZW2EWvpBJA’, ‘a5w1QjNqmxAAAAAAGqoVPw==’]	Pass

Test Unit: compareKey tests

Table 4: compareKey tests

Test Case #	Initial State	Input	Expected Output	Result
1	-Global variables initialized -Local variables declared but not initialized.	clean = password123 hash = $2a$10$K86nOX5LUsm/FppRpefo8ADNnx+B+oMlXXXXXGAAAAAA==	False	Pass
2	-Global variables initialized -Local variables declared but not initialized.	clean = password123 hash = $2a$10$K86nOX5LUsm/FppRpefo8ADNnx+B+oMldIhZ0GAAAAAA==	True	Pass

Module Tested: eksBlowfish.js
In this module the global variables declared are:
p_arrays
s_boxes
which consist of arrays of random hexadecimal values generated from Pi. These values were taken directly from the blowfishs (and simplification of Eksblowfish) official website. (https://www.schneier.com/code/constants.txt)

Test File: eksBlowfish/test.js
Test Unit: feistel_cipher test

Table 5: feistel_cipher test

Test Case #	**Initial State**	**Input**	**Expected Output**	**Result**
1	-Global variables initialized -An instance of the eks-Blowfish object	xl = 112888726 xr = -1272277262	Array: [419532600, 26624517]	Pass

Test Unit: feistel_F test

Table 6: feistel_F test

Test Case #	Initial State	Input	Expected Output	Result
1	-Global variables initialized -An instance of the eks-Blowfish object	xl = 579199262	Integer: 2684460832	Pass

3 Non-Functional Requirements Testing

3.1 Usability Requirements:

The usability of the JScrypt project requires the users to have an introductory level of Node.js knowledge. The usability of this project was tested through giving five participants who were new to Node.js a list of tasks such as installing the JScrpyt project for use, running a local server to test features, and testing the encryption methods included in the JScrypt project.

All of the participants were able to install the JScrypt project for use easily with the input of one command, 'npm install'. The 'npm install' command automatically creates the dependencies (node modules) required for the JScrypt project to operate.

Most of the participants were able to start a local node server with ease through entering the command npm start into their shell environment. This was enough evidence to show that the JScrypt project is usable by those with an introductory level knowledge of Node.js. After creating the local node server, users were able to access the contents of our graphical user interface through entering the address localhost:3000 on their default browsers (Safari Version 9.0.1, Google Chrome Version 47.0, and Firefox Version 42.0).

The JScrypt project provides a graphical user interface (GUI) to users after starting their local servers and the users who managed to start their local node servers had little problem in interacting with the JScrypt functions such the hashKey, and getComponents methods.

3.2 Performance / Speed Requirements:

The performance and speed of the JScrypt project is designed to intentionally operate slowly. The encryption algorithm we used (eksBlowfish) is very resource intensive and this is meant to deter hackers from attempting to unhash information through brute forcing the algorithm. Increasing the cost (number of rounds) would also increase the amount of time required for the hashing process.

The following table outlines a test created to find an approximate time hashing requires based on the number of rounds. In this test, the string 'dog' was hashed and the approximate time is derived from the average of 10 runtimes of JScrypt.

Table 7: Cost versus Time

Number of Rounds	Approximate Time (seconds)
10	0.2303
11	0.3569
12	0.5533
13	1.0157
14	1.8059
15	3.4548
16	6.1625
17	12.5335
18	23.8967

3.3 Robustness:

The hashing method included in the JScrypt project only supports hashing inputs of strings with any number of characters between 1 and 51. If an input with 0 characters or an input with greater than 52 characters was given then the string provided will not be hashed. Also, if the string defined was incorrectly inputted by the user then the string hashed would not be the same string the user wishes to have hashed. When comparing the incorrectly hashed string with a correct raw string, the result returned would show that the strings are different due to the human error involved.

3.4 Operational and Environment:

The JScrypt project is designed to work on the official supported browsers of the Node.js JavaScript framework. The JScrypt project can operate on any operating system (Windows, OSX, Linux, Android, iOS, and more) as long as the operating system has access to one of the supported browsers capable of compiling Node.js. Internet Explorer, Firefox, and Safari are capable of running Node.js but Node.js is optimized to be run on Google Chrome because the V8 JavaScript Engine which the language relies on to compile is optimized for Google Chrome.

4 Testing Summary

4.1 Changes Summary

Throughout the construction of the automated testing and manual testing, there were some issues with program that were found, and required changes to be made to the code. The majority of these issues were related to input checking, since some of the functions were not checking all possible boundaries of the input before continuing its functionality.

4.2 Changes Implemented

- Added more checks to the input of getComponents in order to verify that the input hashKey is not null, or is constructed with an invalid format. If either are true, getComponents should return an empty array to signal it was not able to extract all components of the hash key string.

- In hashKey, a check for null on the key string was implemented. Due to a test case failing on null inputs.

4.3 Traceability Summary

1. **Treaceability to Modules**
   Please refer to this documents 'Functional Requirements Testing' section for the traceability of the modules.

2. **Traceability to Requirements**
   Please refer to the Software Requirements Specification Document revision 0 (Section 9) for the corresponding requirement numbers

Table 8: Traceability to Requirements

tests	req1	req2	req3	req4	req5
generateRandomSalt	✓	✓	N/A	N/A	N/A
hashKey	✓	✓	N/A	N/A	✓
getComponents	N/A	N/A	✓	✓	N/A
compareKey	N/A	N/A	✓	✓	N/A
feistel_cipher	N/A	N/A	✓	N/A	N/A
feistel_F	N/A	N/A	✓	N/A	N/A

4.4 Code Coverage Summary

While following a white-box testing approach for constructing the automated testing, the test cases were designed in order to have statement, branch, and conditional coverage. Though, we did not have function coverage since we believed testing would be more accurately constructed by look at each specific function by itself in solitary. For example, in the JScrypt.test file , each test cases for getComponents were specifically designed such that all the if statements were executed (for statement coverage).

5 Revision History

Table 9: Revision History

Date	**Revision #**	**Authors**	**Description**
September 21	0	All members	A Problem Statement
September 28	0	All members	Proof of Concept Plan
October 5	0	All members	SRS
October 19	0	All members	Test Plan
October 26	0	All members	Proof of Concept Demonstration
November 2	0	All members	Design Document
November 9	0	All members	Revision 0 Demonstration
November 26	0	All members	Test Report
November 28	1	All members	Iteration to revision 1
November 30	1	All members	Revision 1 Demonstration
December 8	1	All members	Revision 1 Document