Department Of Computer Science

SHA-3 Visualisation

Author:
Ross McQuillan

Supervisor:
Prof. Nigel Smart

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering

April 28, 2015

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Ross McQuillan, April 2015

Abstract

The first cryptographic hashing algorithms date back to to the late 70's, although not until the 90's did the popularity of hashing functions increase enough to produce the first widely used algorithms, MD5 and SHA-1. Cryptanalysts later found serious security weaknesses in these hashing functions which forced cryptographers to look deeper into the research of hashing algorithms. NIST later would announce a competition to produce a new hashing algorithm, which would become known as SHA-3.

This thesis looks further into SHA-3 and looks to produce an application which could be embedded into web-pages to further help understand and demonstrate how SHA-3 works. The application will be designed to give the user a deep understanding into SHA-3 rather than just a vague idea of what the algorithm does. Not only will this tool be useful for the understanding of SHA-3 but it will also give cryptanalysts a tool to further analyse SHA-3.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Since the start of the computer era, cryptography has become an ever more important piece of the world we live in. Although unknown to many, cryptography is performed throughout tasks we now take for granted. Gaining access to your 'electronic cash', e-messaging or simply viewing a web-page, all use some form of cryptography.

Many hope for a world where security and privacy is no longer a worry, but everybody knows this is far from a reality and that will almost certainly remain the case. Cryptography is therefore a field which needs to continue to advance forward. This requires careful analysis of future and present standards. It may seem contradictory, but one of the best ways to do this is to attempt to break, or find weaknesses in, existing standards. This requires new tools to help cryptographers analyse current and upcoming standards.

Not only is cryptography a difficult subject to study, it is evolving at an increasing rate, leading to problems when educating people about specific standards in cryptography. When new standards are released, the take up rate for them is extremely slow, predominantly because of a lack of trust towards the standard. Trust for the standard comes over time and the more programmers that understand the standard, the more likely they will be to include standards in small applications and thus begins to build trust of a standard within the computer science community.

SHA-3 was a standard introduced in early 2015, meaning very few people understand how SHA-3 works and even fewer trust the algorithm, which is yet to be tested in real world applications. This gives rise to one of the main reasons to develop tools around such an algorithm; to ensure that computer scientists remain educated and up to date in fields which defends sensitive data and allowing them to analysis the algorithm for themselves.

1.2 Statement Of Problem

Currently, SHA-3 has few tools available to computer scientists which aid in the teaching of the algorithm. This would be very beneficial to lecturers and students, significantly decreasing the time needed to explain the algorithm and then helping students during self-study of the algorithm. Not only is there a lack of tools in teaching the algorithm but also for analysis of SHA-3 and these tools will become ever more necessary when SHA-3 begins to gain momentum.

1.3 General Aims

The general aim of this project is to develop an application aiding in the teaching and analysis of SHA-3. The application will be able to calculate SHA-3 and breakdown each of the stages in the algorithm for the user, whilst also including tools which will allow for a better analysis of the algorithm. Above all, the tools provided must take a different approach to pre-existing software, which will be done by visualising the algorithm (explaining how we will visualise SHA-3 will be covered further into this thesis).

Primarily this application will be aimed towards people looking to learn how SHA-3 works. Alongside this group of people, cryptographers looking to analyse the algorithm will benefit greatly from the application.

The application must also be easily accessible for all who wish to use it, meaning not only does the application have to be cross platform but it also must run efficiently as possible, for users who require to run the application on older machines.

1.4 Outline Of Thesis

This thesis will be structured in a logical order such that the reader may follow the project through each of its stages and obtain a good understanding as to how each choice was made and how the development of the application occurred. Each remaining chapter contains the following information:

Chapter 2 : This chapter will discuss the background of the project and the research that took place. It will look into other applications which may already exist with the same objectives, it will argue the reasoning behind the decisions and it will examine other research.

Chapter 3 : This chapter will explain any decisions taken prior to developing the application, before going on to discuss the specification and the design of the application.

Chapter 4 :

Chapter 5 :

Chapter 2

Background And Research

This chapter will look at explaining the neccessary requirements to understanding the project. It will explain the functionality and applications of cryptographic hashing algorithms, before looking deeper into SHA-3, explaining the algorithms origin and how it works. Finally, it will review similar work published by different authors and the software that has been developed alongside of it.

2.1 Cryptographic Hashing Algorithms Overview

2.1.1 Brief History

The first of many cryptographic hash functions was created in the late 1970's but the topic didn't gain any noticeable momentum until the early 90's when the first widely used algorithms became popularised. These algorithms are known as MD5 and SHA-1 and were the first cryptographic hashing algorithms to be included in a wide number of applications.

Throughout the start of the new millennium, cryptographers found serious security flaws in both, MD5 and SHA-1, forcing significant progress to be made in the field. SHA-2, which had already been created prior to these weaknesses being discovered, had many similarities to SHA-1 but the same weakness could not seem to be used to exploit the SHA-2 algorithm in the same way.

After the weaknesses in SHA-1 and MD5 became exposed, NIST (National Institute of Standards and Technology) announced a competition to create the next member of the SHA family, SHA-3. It was decided that the algorithm should be designed separately to SHA-1 and SHA-2 in case further weaknesses were yet to be exposed. In 2013, NIST began the standardisation process for the algorithm and in late 2014, they published a draft of SHA-3.[1]

2.1.2 Functionality

A cryptographic hash function has one primary goal, which is to be used as a map from some input string, the message, to an output string of a fixed length, the hash value. To ensure the hash function is secure, it is required that it is "practically impossible" to invert in addition to having a very low probability of 2 input strings producing the same output string. In other words, given a hash value it would be "hard" to discover what message generated such a value. A more precise definition is used by cryptographers which will be examined more closely further into this thesis. Difficulty, such as uses of the word "hard", can be quantified.

2.1.3 Applications

Cryptographic hash functions have a wide range of uses in computer science including password verification and pseudorandom generators.

Password verification

Clearly, storing all passwords in plain text is a bad idea, since if somebody gains access to the database they immediately have everybody's password. A widely used and trusted alternative to this is to use a process called hash and salt, where as the password is created or changed you immediately apply a hash algorithm to the password and store this instead. Then when the user comes to log into their account the next time you can apply the same hashing algorithm to the entered password and compare this to the password stored in the file. This is still very insecure, so for added security a randomly generated string, called a salt, is concatenated to the password before the hashing algorithm is applied, then stored alongside the hash value for future references.

Integrity Of Data

Checking to see if a message or file has been altered is an important process which must be completed. One way to achieve this is to calculate the hash value of the data. By taking the binary representation of the data, we can apply the hash algorithm to this string and return the hash value. Then when the message is sent to the recipient, the hash value is sent alongside it, both of which are encrypted. Upon receiving both, the recipient applies the same hash algorithm to the message and compares this with the hash value. If they match, the recipient can confirm with a high degree of certainty that the data has not been altered in transmission.

Pseudorandom Generators

Hash functions can be used to generate pseudorandom strings of a fixed length, which can be used as a private or public key for an encryption system. Giving the hashing algorithm different messages, or seeds as they are more commonly known, when the hash algorithm is being used as a pseudorandom generator, gives different unpredictable outputs. Each output can be considered to be a different random number.

2.1.4 SHA-3

SHA-3 is the hashing algorithm which will be looked at in much more detail in this project. Therefore a more in-depth explanation of what SHA-3 is will need to be given.

SHA-3, like other hashing algorithms, takes an input and outputs an unpredictable, irreversible string of a fixed, predetermined length. It does this by taking the input string and padding this string to a required length. This string is then XORed into the state.

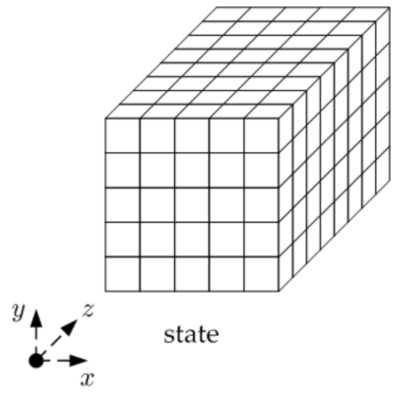


Figure 2.1: SHA-3 State

The aforementioned state is a bit string represented in a specific format and is represented as a 3-dimensional array of size $5 \times 5 \times 64$, initialised completely to 0 at first. Each element of position (i, j, k) can therefore be accessed easily with state[i][j][k]. An illustrative example, although smaller dimensions, of the state is given in figure **??** [2].[1]

To absorb the input string into the state, we take the first r bits of the string, where r is predefined, and XOR these bits with the first r bits of the state. Next, the block permutations are applied, transforming the current state into an alternative state. This will be explained in a greater detail in the following paragraph. If any bits remain in the input string, the process is repeated with the next r bits until the string is completely absorbed, or alternatively until we are left with the empty string. This type of structure is known as a sponge construction.

The block permutation is made up of 24 rounds, and each round is almost identical to every other. Rounds are made up of 5 sub-rounds which, when applied to the state, transforms it in some way.

The first 4 sub-rounds, named θ, ρ, π and χ, follow the same algorithm identically each time, whereas the final permutation, ι, relies upon which round is currently being completed. When these 5 sub-rounds are performed one after the other, in the order given, the algorithm is said to have completed 1 round. For each block permutation to be completed 24 rounds are completed in succession.

Appendix A gives a more mathematical, in-depth explanation of SHA-3.

2.2 Previous Research

After extensive research into the field of study, no discovery has been made of another piece of software which offers a solution to the problem which has been set out by this project. However, similar visualisation applications exists for both SHA-1 and SHA-2. One of the most popular papers produced, named SHAvisual: A Secure Hash Algorithm Visualization Tool [3], created a visualisation tool for SHA-512 to aid both

[1]A state has size $5 \times 5 \times 64$ if the 'complete' SHA-3 algorithm is being used. Although, for lighter weight applications a smaller state may be used providing the size is of the form $5 \times 5 \times 2^l$ where l is some integer less that 6. Further information surrounding this is discussed in the appendix.

students and instructors in the learning and the teaching of the secure hashing algorithm. The algorithm demonstrated the major components of the algorithm in a step-by-step structure, whilst also providing a simplified version to be used in presentations. The tool also offers a ‘full mode’ which can be used to perform full SHA-2 hashing.

Another paper published focused more closely on SHA-1 and creating an applet which looks into the weaknesses and strengths of SHA-1, whilst also producing an “interactive visualisation for teaching SHA-1”. The paper, Visualizing Secure Hash Algorithm (SHA-1) on the Web [4], looks to gain knowledge about the algorithm, SHA-1, and then further looks to evolve the applet for use with SHA-2. This project looks to implement a similar application, to be embedded into the web for use with SHA-3.

Chapter 3

Specification and Design

This chapter will focus on examining the specific details required for developing a successful application. It will look closely at decisions which need to be made prior to the implementation of the application. It will continue to explain how SHA-3 will be visualised and discuss how this will be presented to the user. This section of the project is vital to ensuring the implementation of the project goes as smoothly as possible.

3.1 Project Scope

In section 1.2, the outlining problem that this project looks to solve was stated. In this section we look at setting goals which they project will aim to achieve that will solve the problem previously stated.

3.1.1 Project Goals

The goals that are to be set must focus on producing an application that aids the process of teaching/learning SHA-3 and the analysis of the algorithm. They also must give a decisive objective which can be worked towards. These goals will help measure the success of the project alongside user feedback.

When producing the goals, a well thought through and methodical process was established, to ensure that the goals are achievable and measurable. From such, the following goals were concluded:

- Develop an ease of use interface to ensure users are not dissuaded from using the software.
- Give a simple, intuitive visualisation of the state.

- Breakdown SHA-3, using it's sub-rounds, making it easier to explain.
- Provide the user with a detailed explanation of SHA- to accompany the application.
- Supply tools which can be used to help further analyse the algorithm.

3.2 Decisions

Prior to designing the application, decisions must be made regarding the progamming language and 3D libraries. This careful planning is to ensure that the chance of backtracking in the development phase is reduced to a minimum, whilst also giving the developer a clear idea of how the project will be developed.

MAYBE OVERKILL ON Programing Language SECTION! TBD

3.2.1 Programming Language

There are 3 major programming languages with which the application can be developed with. These include JavaScript, Java and C++, each of them coming with their positives and negatives.

JavaScript

The main advantage to using JavaScript would be the ease of access to all users. JavaScript can easily be embedded into any website and viewed on any web browser without the need for any additional add-ons. Furthermore, JavaScript has a wide range of libraries which are accessible, with documentation covering all areas of the language.

Although JavaScript looks to have many strong qualities, one drawback is that it is not a programming language which has been used previous to this project in other applications. The means that the project would take much longer than necessary to complete as JavaScript would be learnt whilst developing the application.

Java

Like JavaScript, Java gives the most users possible access to application, since Java can also be embedded within a web page. Although additional tools and software are required to run the embedded applet. Meaning getting a applet on a website to run can soon become much more complicated process for the user.

Java is one of the most extensively covered programming languages available, with clear documentation for all accessible functions. Alongside this, Java has many libraries available to it, which have been developed over the 20 years which it has been accessible.

Finally, Java is one of the most powerful programming languages available. Giving developers access to objects and classes and thus allowing an easier development of an application.

C++

C++ is one of the most commonly used programming languages for developing 3D worlds. Like both Java and JavaScript there is extensive documentation available to developers and also a wide range libraries for developing 3D graphics.

Since C++ is primarily used in the development of games, meaning the libraries available contain much more resources that what is necessary in the development of this application. The major concern with C++ is the lack of portability of the final application, since it can not be effectively implemented inside of a web browser.

After looking at the pros and cons of all the programming languages, one programming languages stand out above the others. This programming language is Java because of its portable and powerful nature.

3.2.2 3D Graphics Package

After selecting Java as the programming language, the libraries which are to be used must also be selected. There are a few packages available to Java, which have the ability of producing 3D graphics. Of which, the project requires that the library must:

- Be well documented.
- Have the ability to be embedded within a browser.
- Be simple and easy to access.

Given these requirements, Java 3D[5] is the well documented and easily accessible library which will be used in the application.

3.3 Visualising SHA-3

The way that this project will visualise SHA-3 will be using the state, which, as mentioned in the previous chapter, is represented by a 3-dimensional array. To break down the visualisation further, we require that each element of this array has a visual representation, for which we use a cube, using the colour of the cube to show the value of the bit stored in the element. This then allows the entire state to be created using multiple cubes organised in a specific way which equates to the state.

This representation of the algorithm allows for much more than the simple analysis of the hash value. Instead it allows the users to display midpoints in SHA-3 which previously could not be viewed. This therefore already gives the user the ability to further analyse SHA-3, before the inclusion of any tools specifically designed in aiding the analysis of the algorithm.

3.4 Tools

This section looks at the tools which can help cryptographers further analyse SHA-3 but more importantly looks at teaching and furthering the understanding of the algorithm in people looking to learn more about it.

3.4.1 Individual Sub-Rounds

To give an in depth understanding of SHA-3, the fundamentals must be firmly grasped by the user. The fundamentals of the algorithm lie within each of the sub-rounds, meaning if the sub-rounds are understood correctly then the complete SHA-3 algorithm will soon follow.

This tool will provide the user with example of each sub-round being applied to a state. The initial state must not be complex, as this will look as if the state has been randomly changed when the sub-round is applied. Instead the state must contain mostly 0's, which will show clearly any bit changes in the state and help to avoid confusion.

3.4.2 Cumulative Difference

Cumulative difference runs SHA-3 on a state, in the exact same way as usual. The tool then analyses SHA-3 as the algorithm is running, calculating the differences from a current state to a new state each time one of the sub-rounds are applied. In other words, the tool generates a new state with the value of each element being the differences between the two states, one prior to the sub-round being applied and the other, after the sub-round was applied. The tool then sums all of these newly generated states, creating the state representing the cumulative difference.

The user will be able to move through the states visualising the cumulative difference up to a specific point in the algorithm. This comes with a problem when visualising the cumulative difference which will also be represented as a state. A state usually holds a single bit which can easily be represented on a cube as colouring the cube black or white, although values stored for the cumulative difference are integer values not bits. Therefore visualising this state will be done by supplying each cube with a colour scaled between white and red dependent upon its value in the array. Meaning, the further along in the algorithm which the user look at the deeper the colour of red which will appear. This is be each bit will have changed more often.

Before using the tool, one may expect that the outcome of this method produces a state with each element containing similar values. Equivalently, each bit can be thought of as changing an equal number of times throughout the algorithm. If so this tool will prove the strength of the algorithm. Although it is a possibility that the cumulative difference is dependent on the initial state and thus maybe exposing weakness in the algorithm which could be exploited.

3.4.3 Random Bit Difference

One of the main benefits to visualising the gradual progression of SHA-3, is to allow, for educational purposes, the user to view how quickly two similar initial states, with only one bit difference, diverge from each other. This tool, random bit difference, shows this attribute by taking 2 initial states, where the second state is created from the first with only one bit difference. It then applies SHA-3 to both and records the state prior to each sub-round and finally the state once the algorithm is completed. The tool then generates a set of states which has been calculated from the two recorded sets. This is done by taking the difference of two states, one from each recorded set, at equivalent points throughout the algorithm.

This tool, as mentioned previously, provides the user with a visual representation of how quickly two similar states diverge from each other. Generating very different paths from one another before finally generating two completely different hash values; essentially the main purpose of a cryptographic hashing algorithm.

3.4.4 Random Bit Difference Summation

The random bit difference summation tool, uses the previous tool and adds to it, an extra layer which can be used to analysis SHA-3 further. Similar to the previous algorithm it produces the first state, the original state. From this state multiple different states are then generated, with again only one bit difference, ensuring that no two states are the same. SHA-3 is then applied and before each sub-round is applied a summation of all the states is calculated and stored separately. After the final sub-round is applied and SHA-3 has completed, then the final summation is calculated and stored.

Similar to the cumulative difference, the state containing the summation of all other states will not have a single bit stored in each element of the array. Instead an integer will be stored and this will be represented in the same way which the cumulative difference tool handles this. This is done by assigning a colour between white and red, with the deepness of red representing the value of the element in the array.

This tool gives analysts the ability to study the probability of certain outcomes occurring when changing specific bits and allows them to try and predict the the changes that will occur in the state. If this then becomes possible it exposes serious weaknesses in the algorithm.

3.5 Graphical User Interface

This section will discuss how the user will interact will the application, such that a well thought out interface is developed, giving the user the best possible experience available.

3.5.1 Menu

The application will be structured such that it focuses on simplicity, allowing the user to focus entirely on the SHA-3 algorithm. The best way to do this would be to have the application open with a simple menu, listing options to the user.

There will be 4 options applicable to the user:

- Standard SHA-3
- Individual Sub-Rounds
- Cumulative Difference
- Random Bit Difference
- Random Bit Difference Summation

When selecting any one of these options they are expand into a new window, displaying the tool. To avoid confusion and allow for a simpler experience, the initial state is randomly generated, meaning the user does not have to worry about preconfiguring the tool. The only choices the user will have to make involve selecting which sub-round they wish to view and also the size of the state.

3.5.2 Interaction

Visualising a state in a simple and easy way is critical to ensuring the application succeeds. The user must therefore be able to view the state from different sides and angles. This means the application must be interactive. Giving this ability to the user means that they are able to manipulate the cube to view different angles and directions which gives a greater sense of immersion.

This will be done by allowing the cube to be rotated and moved using the mouse by clicking and dragging, giving the user a similar feel to as if the state was within their hands. Using the arrow keys on the keyboard, the user will also be able to zoom the state in and out viewing more or less of the state.

Navigation bars must also be provided alongside the view of the state, to allow the user to move to and from different stages within each tool. The navigation bar will contain simple arrows directing the user to the first, previous, next and last stage in the SHA-3 algorithm. The navigation bar will also have a 'go to sub-round' option which allows the user to entire a value and move directly to that stage.

3.6 Technical Difficulties

The SHA-3 algorithm has been implemented many times, in many different programming languages. At the time of the research for this project, all implementations of the algorithm which were discovered, are in many programming languages other than Java. Meaning that SHA-3 must be re-written into Java or ported from another programming language which has the ability of being implemented into a web page.

After closer analysis of these implementations of SHA-3 and of the source code provided on the Keccak website[2] I discovered that they each uses the same implementation method. This, whilst logical, is not beneficial to the project since the algorithm has been coded for efficiency and as such multiple sub-rounds of the algorithm have been merged together making it impossible to view a state between these sub-rounds.

To solve this issue, it is required that SHA-3 is understood in depth and only then can the fundamental, mathematical algorithms for each sub-round can be re-written into Java. Once this has been produced, the well commented and simplistic code can then be used in the teaching of

SHA-3, alongside the application itself.

A second issue which must be dealt with, is when using the complete SHA-3 algorithm the state becomes very large and as such can become overwhelming and confusing to a user. Therefore a simple and elegant solution must be found, although without first viewing the state and solution can not yet be reached as this could be come a much bigger problem than originally though. Therefore, this issue will be discussed again in the next section when a further understanding of the issue has been gained.

Chapter 4

Implementation

This chapter will discuss the process in which the application was created and will look at each individual step within this process. There are 4 main areas to be discussed. Firstly, generating a basic 3D world, then building on this to produce a representation of the state which allows user interaction. The next task, which is independent to the first, would be to produce the SHA-3 algorithm. It will then look at producing the specific tools for the application, as previously mentioned in chapter 3. Finally, it will look at implementing the user interface, with the navigation keys and menu.

The reason for breaking a large project down into stages, is to allow each stage to be tested individually and to allow for any bugs to be caught early on in the development. Therefore isolating the bug to a specific area and making the solution of any problems much simpler. Although when developing like this it does introduce new problems such as compatibility. Therefore, ensuring that when developing two individual systems, which are to be merged later, it is important that the same format for shared data is used, making merging the systems much easier.

All of the objects and Java3D terminology in this chapter will be discussed in the level of detailed required to gain a good understanding, but for more information visit the Java3D documentation page[6].

4.1 3D World

Developing a 3D world, containing a visual representation of a state, is the foundation of this application, meaning this is the best place to start developing the application. The first task is to develop a “universe”, using the Java3D library . A “universe” contains the canvas, which is where all the 3D objects are drawn.

The CreateCubes class is where the 3D world is produced. It uses pre-defined static variables to set the width, depth and height of each cube in the state, where as the users initial view of the state varies, and is calculated, dependant upon the size of the state in use. This is to ensure a better user experience when dealing with larger states.

Directional lighting is produced within this class, to give the user a sense of depth when visualising the state. The lighting is implemented such that the light souce appears to be just off screen. The positioning of the lighting is set with a 3D vector, vector3f, and the correct positioning is essential to making the viewing of the state ease on the eye.

The class also holds two 3-dimensional arrays. The first, “cubes”, is used to control the cubes which are used to visualise state. Each element stores a single cube, with the appearance of that cubes also kept alongside it, making it vary easy to change the colour, shading and lighting of each cube. The second array, “currentState”, stores the current values of the state. The values stored in this array is dependant on what the visualisation is required to show. For example, if the state was required to display SHA-3 then the values stored will be either a 1 or 0.

Each 3D array is indexed such that access the same point in each array refers to the same point in the state. This makes updating the cube much easier, as when the values of the state need to be changed, it is simple to also update the cubes appearance at the same time.

Figure 4.1 shows the current representation of the state, if populated with random bits, where black cubes represent a bit value of 1 and white cubes represent a bit value of 0.

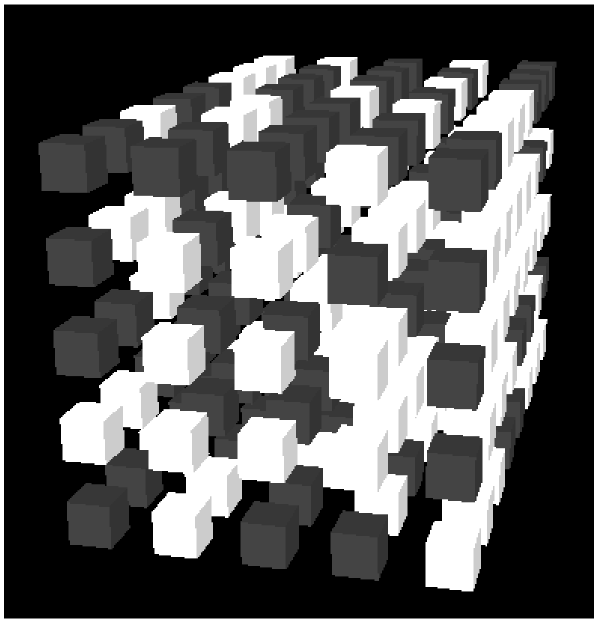


Figure 4.1: Visual Representation of the State in SHA-3

4.2 SHA-3 Algorithm

The SHA-3 algorithm is developed in its own class, "SHA3", and the class consists of 7 functions

The first 5 functions are the sub-rounds, Theta, Rho, Pi, Chi and Iota. Each take in the 3-dimensional array, the state, with the exception of Iota, which also takes in an integer, the current round. The output of each function is the state, returned in the same format as it was input, a 3-dimensional array.

After completing each of the sub-rounds, it becomes very simple to produce the 6th function, a complete single round of SHA-3, by simple running each sub-round sequentially in succession. Then repeating each round the required number of times, 24 for the complete SHA-3 algorithm, produces a complete algorithm. The complete SHA-3 function can be seen in figure 4.2.

The final function takes in a state as input and outputs a string. This function represents the final stage of SHA-3, returning a hexadecimal string, the hashed value, of the input.

```
SHA3(int[][][] state) {
  for (int i = 0; i < rounds ; i++) {
    state = theta(state);
    state = rho(state);
    state = pi(state);
    state = chi(state);
    state = iota(state, i);
  }
  printHash(state);
};
```

Figure 4.2: Complete dunction for SHA-3

Problems

When producing the algorithm in the Java programming language, taking an implementation of the algorithm already produced in not a possibility. As previously mentioned, other implementations merge the sub-rounds together into one concise function.

Alongside the aforementioned issue, other implementation can sometimes be difficult to understand what is occuring in each of the sub-round functions. Functions which allow for an easier understand could also be used in the explanation of SHA-3 and be much more beneficial to the student/tutor. Both of these points are easily demonstrated in figure 4.3.

Understandable in doing this efficiency and neatness is compromised, but these are not the main aims of the project therefore they are a necessary compromise.

The Keccak website[2] and the SHA-3 wikipedia page[7] both have documentation which give a broken down implementation of SHA-3. Using this documentation, each subround was re-created in turn, in it's most simple form for understanding and seperating out each sub-round in turn.

It must be noted that while the Keccak website and the SHA-3 wikipedia page provide extremely useful information, they do contain conflicts in notation and one must be careful when using multiple sources to check for any conflicts and avoid confusion.

```
rho(int[][][] state) {
  int newState[stateWidth]
    [stateHeight][stateDepth];
  for(x=0; x<stateWidth; x++) {
    for(y=0; y<stateHeight; y++) {
      for(z=0; z<stateDepth; z++) {
        newZ = (z +
          rotation_const[x][y])
          % stateDepth;
        newState[x][y][newZ]
          = state[x][y][z];
      }
    }
  }
  return newState;
};
```

(a) Application's Implementation of ρ *in Pseudocode*

```
pi(int[][][] state) {
  int newState [stateWidth]
    [stateHeight][stateDepth];
  for( x=0; x<stateWidth; x++) {
    for( y=0; y<stateHeight; y++) {
      for ( z=0; z<stateDepth; z++) {
        newX = y;
        newY = (2 * x + 3 * y)
          % stateHeight;
        newState[newX][newY][z]
          = state[x][y][z];
      }
    }
  }
  return newState;
};
```

(b) Application's Implementation of π *in Pseudocode*

```
temp = state[1];
for (x=0; x<24; x++){
  BC[0] = state[KeccakF_PiLane[x]];
  state[KeccakF_PiLane[x]] = ROL(temp,
    KeccakF_RotationConstants[x]);
  temp = BC[0];
}
```

(c) Keccak's Implementation of both π and ρ *in C*

Figure 4.3: SHA-3 Sub-rounds ρ and π

4.3 Tools

This section will look at the implementation of each of the tools mentioned in the previous chapter.

4.3.1 Visualising SHA-3

After completing the visualisation of the state and the SHA-3 algorithm, the next step would be to merge the two implementations to give one working system.

From acknowledging a future problem, the compatibility of the two components, early on in the implementation, it has made the process of merging the two separate parts into one much easier. This is because the creation of the state in the CreateCubes class is of the same format which is to be entered into SHA-3 and also the same output of SHA-3. This means when the state is randomly generated, it can be immediately passed to the SHA-3 algorithm and an array of states will be returned, with each element containing the state at single unique points in the algorithm. Now any element in this array can be selected and displayed using the CreateCubes class, showing how the state progresses throughout SHA-3.

4.3.2 Sub-Rounds

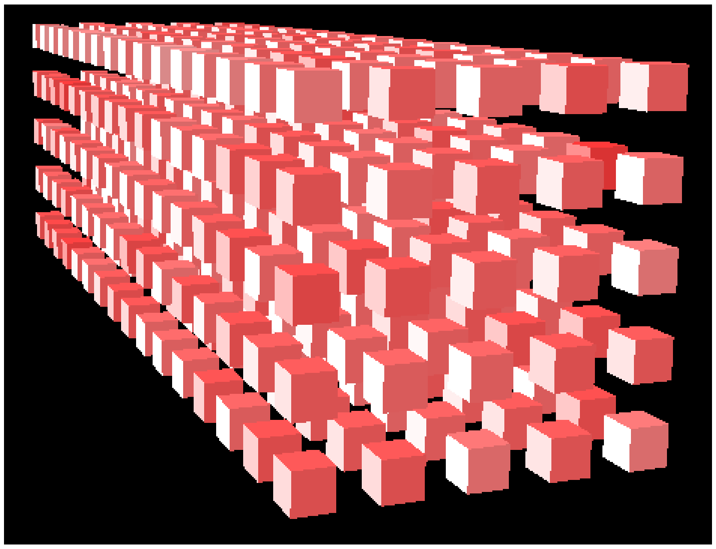
Like the full visualisation of SHA-3, individual sub-rounds are equally simple to implement. By repeatedly calling one of the 5 sub-rounds in succession, one can again obtain an array which contains the states after each sub-round is applied.

This tool is produced in the class "SubRounds", and although it is only a small segment of code, placing it in its own class gives greater flexibility when selecting specific starting states, navigation options and gives better organisation to the code.

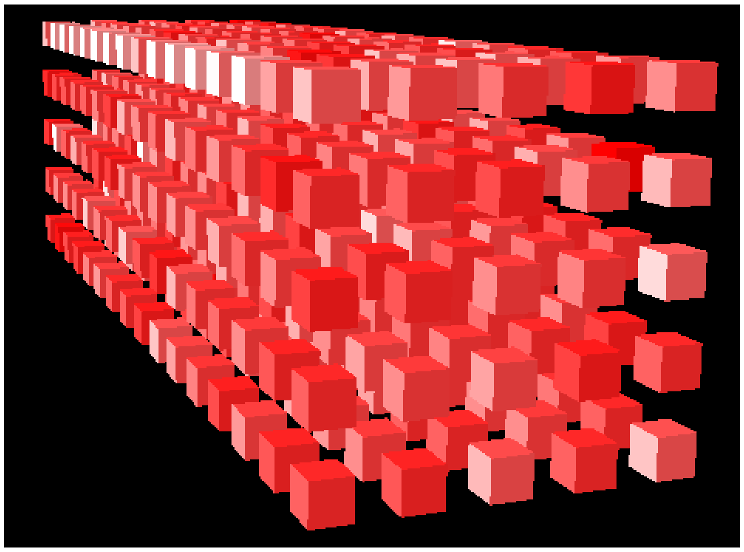
4.3.3 Cumulative Difference

Generating the "CumulativeDifference" class requires more calculations than that of either the SHA-3Visualisation or the SubRounds class. The idea of this class is to develop a tool which calculates and visually shows how often each bit, in each position of the state flips, or changes from a 0 to a 1 or alternatively from a 1 to a 0. It is visualised in the CreateCubes class as a deepness of red, the redder a cube the more times it's value has changed. In figure 4.4 it is easy to see the difference between the two states at two different stages in the algorithm.

A more in-depth analysis of these states will be given in the results chapter, as one may notice in figure 4.4b an interesting property of the cumulative difference.



(a) Cumulative difference calculated for each bit in the state, mid-way through completing the algorithm.



(b) Cumulative difference calculated for each bit in the state, once SHA-3 has been completed in its entirety

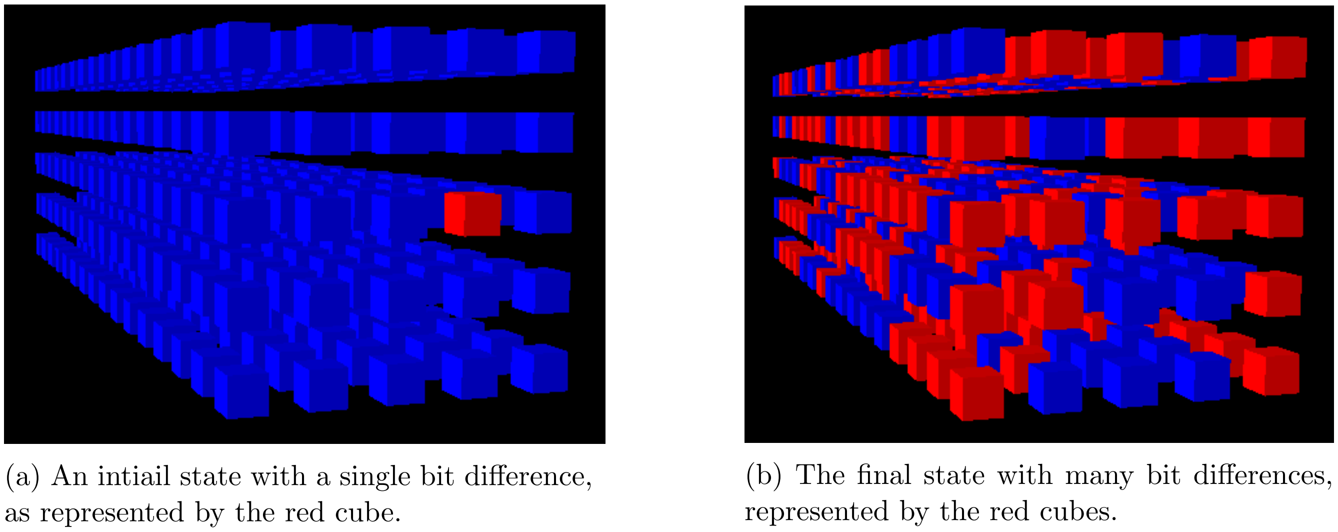
Figure 4.4: Shows the cumulative difference at different point throughout the SHA-3 algorithm

The cumulative difference class first runs the SHA-3 algorithm on the initial state, storing the state at each point throughout the algorithm. This process is identical to the one used for the SHA3Visualisation class and is therefore logical to use this class to do this first step. The cumulative difference tool then goes a step further to begin to analyse the SHA-3 algorithm. This is done by taking the returned array of states from SHA3Visualisation, running through them and for each bit calculating the summation of the number of changes each bit has gone through. These summations are then stored into a state themselves and can them be used to display the cumulative difference in the CreateCubes class.

4.3.4 Random Bit Difference

Random bit difference, as discussed in the previous few chapters, looks to show the diffusion property of SHA-3. Again this tool builds on the "SHA3Visualisation" class. Firstly a randomly generated state is created, one bit is then selected from this state and is changed, now giving 2 similar, but unique, states. SHA-3 is then applied to both in turn giving now two very different arrays of states. The main function in this class, "stateDifference", takes in these two arrays of states and computes the difference at each point.

The state difference cycles through each of the arrays one after the other comparing the difference of each bit in the state with the bit of the same position in the other array. This is done simply by XORing the two values together. The function then returns an array of states containing the differences, the initial state containing only one difference and the final containing a 50/50 spilt of differences on average, as shown in figure 4.5.


(a) An intiail state with a single bit difference, as represented by the red cube.

(b) The final state with many bit differences, represented by the red cubes.

Figure 4.5: A clear illustration of the diffusion property in SHA-3

Above, in figure 4.5 it is clear to see how very subtle changes in the initial conditions cause very different end states. Viewing individual steps within the algorithm, it becomes clear to the user how easily one bit change propagates through the state.

4.3.5 Random Bit Difference Summation

Random bit difference summation resides within its own class, “MultipleFlippedBit” , and replicates the previously mentioned tool, flipped bit difference on a larger scale. Instead of creating only one different states with a single bit changed, multiple states are created with a single bit changed instead. For example, instead of starting with 2 initial states the class will use in excess of 20 states, each with only one bit difference from the generator state, the original, randomly generated state. After SHA-3 is applied to all the states, the tool sums the total number of differences between each state and the generator state and stores the values into a a state of its own, which is then returned.

The total number of differences state is then displayed in a similar way to cumulative difference, by using the gradient of a colour to represent how many times a bit differs from the generator state.

Since changing one bit should make the state indistinguishable to the original state after SHA-3 is applied, effectively appearing random to anyone viewing the state. Summing multiple differences between these states will give a final total state which has very similar values in each element, again showing a clear illustration of the diffusion property.

4.4 Navigation Bar and Menu

4.4.1 NavigationBar

The navigation bar resides within each class, as the content of the navigation bar depends entirely on what the user is viewing at that moment and therefore relies on the tool currently being displayed.

Each button within the navigation bar was created using the JButton class provided by Java libraries. This JButton then is positioned within a JPanel which is placed at the top of the window for ease of access.

4.4.2 Menu

The option menu will be the first window to open when the application is launched. It will provide the user with a list of the tools available to them, with the final element of this list containing settings. Settings allows the user to change options such as the size of the state.

Chapter 5

Results

This chapter will look to answer the question of how successful this project has been. It will look at two different ways of measuring this. Firstly, returning to the project goals, stated in the specification and design chapter, and discussing whether each goal was a success or failure. The second analysis of the project will be achieved using user feedback. This will be done by allowing them to use the application and asking them specific questions surrounding their experience, whilst also asking users to compare learning SHA-3 with the application to without it.

References

[1] NIST (National Institute Of Standards and Technology). The SHA-3 Cryptographic Hash Algorithm Competition. `http://csrc.nist.gov/groups/ST/hash/sha-3/`, 2015.

[2] Bertoni, G. and Daemen, J. and Peeters, M. and Van Assche, G. The Keccak sponge function family. `http://keccak.noekeon.org/`, 2015.

[3] J. Ma, J. Tao, M. Keranen, J. Mayo, C.-K. Shene, and C. Wang. SHAvisual: A Visualization Tool for Secure Hash Algorithm, 2014.

[4] D. Nasr, H. Bahig, and S. Daoud. Visualizing Secure Hash Algorithm (SHA-1) on the Web, 2011.

[5] Wikipedia contributors. Java 3D. `http://en.wikipedia.org/wiki/Java_3D`, 2015.

[6] Java 3D Parent Project — Project Kenai. `https://java3d.java.net/`, 2015.

[7] Wikipedia contributors. SHA-3. `http://en.wikipedia.org/wiki/SHA-3`, 2015.