

Assignment 1

The Avalanche Effect



September 26, 2020

Sukhman Atwal

Contents

[Introduction 2](#_Toc52003193)

[How to Use 3](#_Toc52003194)

[Design 4](#_Toc52003195)

[Diagrams 5](#_Toc52003196)

[Testing 7](#_Toc52003197)

[Test 1 8](#_Toc52003198)

[Test 2 9](#_Toc52003199)

[Test 3 10](#_Toc52003200)

[Bibliography 13](#_Toc52003201)

# Introduction

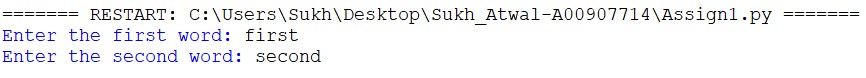
The objective of this assignment was to use a cipher of our choice that would read hashes generated by the tool and perform a bit-by-bit comparison, producing count value(s).

For the purposes of this assignment, I used SHA256 (Secure Hash Algorithm 256-bit) which was designed by the NSA and first published in 2001.

I created a script using Python 3 to convert two user inputted strings to sha256 hash and to binary. The script would then compare the binary outputs and count the difference between the two. All of this data would then be saved in a csv file.

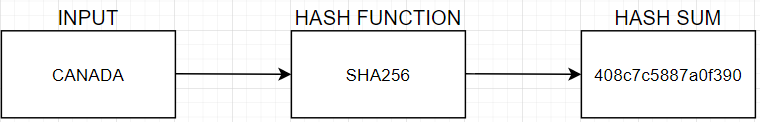
# How to Use

You would run the script using the command line, which should then ask for user input twice.

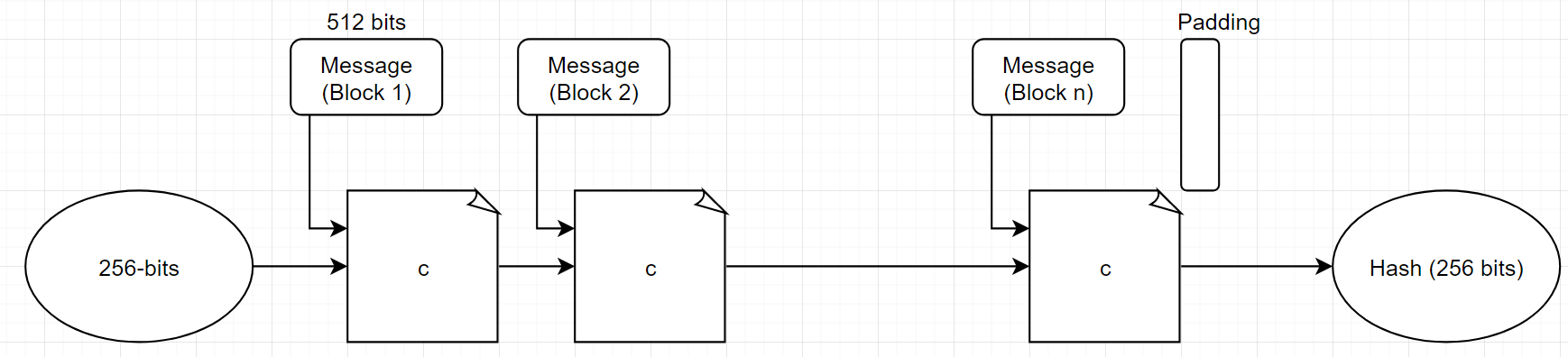


Afterwards, all relevant information will be displayed and saved to a csv file located in the same directory as the script.

# Design



A simplified version of how it works. You enter an input, in this case ‘Canada.’ It goes through using the hash function of your choice, SHA256 here and outputs the hash sum.



SHA256 turns inputs into 512 bits sized messages which are then fed to ‘C’ which is a compression function producing a 256-bit output which would be the input for the ‘second’ message. This continues until the end of the messages. (Kanumarlapudi, 2018)

# Diagrams

I’ll start by showing the changes in bits in the table below for the word ‘test’, changing the letters one by a time.

|  |  |  |  |
| --- | --- | --- | --- |
| Word | Changes in Text | Binary | Changes in Bits |
| test | 0 | 01110100011001010111001101110100 | 0 |
| tess | 1 | 01110100011001010111001101110011 | 3 |
| tett | 1 | 01110100011001010111010001110100 | 3 |
| tets | 2 | 01110100011001010111010001110011 | 6 |
| trts | 3 | 01110100011100100111010001110011 | 10 |
| etts | 4 | 01100101011101000111010001110011 | 10 |

The graph above shows the changes in text and changes in bits over the 6 iterations.

I also did it with the word ‘securities’, changing letters one at a time.

|  |  |  |  |
| --- | --- | --- | --- |
| Word | Changes in Text | Binary | Changes in Bits |
| securities | 0 | 01110011011001010110001101110101011100100110100101110100011010010110010101110011 | 0 |
| securitiea | 1 | 01110011011001010110001101110101011100100110100101110100011010010110010101100001 | 2 |
| securitiab | 2 | 01110011011001010110001101110101011100100110100101110100011010010110000101100010 | 3 |
| securitabc | 3 | 01110011011001010110001101110101011100100110100101110100011000010110001001100011 | 5 |
| securiabcd | 4 | 01110011011001010110001101110101011100100110100101100001011000100110001101100100 | 12 |
| securabcde | 5 | 01110011011001010110001101110101011100100110000101100010011000110110010001100101 | 10 |
| secuabcdfg | 6 | 01110011011001010110001101110101011000010110001001100011011001000110011001100111 | 17 |
| secabcdefg | 7 | 01110011011001010110001101100001011000100110001101100100011001010110011001100111 | 12 |
| seabcdefgh | 8 | 01110011011001010110000101100010011000110110010001100101011001100110011101101000 | 21 |
| sabcdefghi | 9 | 01110011011000010110001001100011011001000110010101100110011001110110100001101001 | 21 |
| abcdefghij | 10 | 01100001011000100110001101100100011001010110011001100111011010000110100101101010 | 24 |

The graph below shows the changes in text and bits over the 11 iterations. We can see some dips taking place for the bits.

Here we specifically look at the bit’s changes over time. We can see an avalanche effect taking place as the changes in bits are rising, falling, and then rising again.

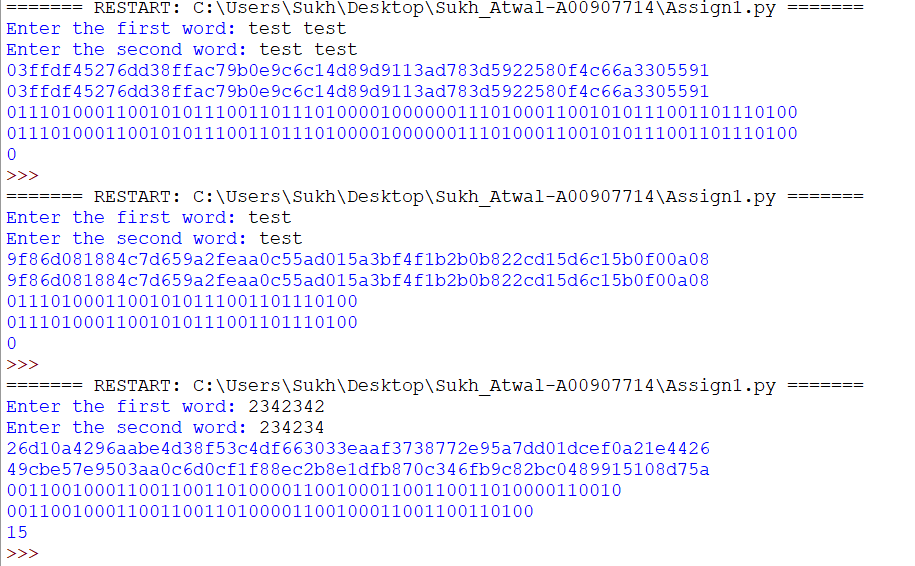
# Testing

Here we are checking to ensure we are getting the correct outputs from the user inputs.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Description** | **Tools Used** | **Expectations** | **Actuality** | **Pass/Fail** |
| 1 | Users are able to enter anything into the user input section | IDLE | The user should be able to enter anything into the user input section and it should display back the hash and binary values. | The user is able to input any word or number with no errors and the hash and binary values are correctly displayed in the console. | Pass |
| 2 | The data is outputted to an csv file | IDLE, CSV | The data entered should be sent to an csv file located in the same directory as the python script. | The same data inputted and seen in the console is also displayed in the csv file located in the same directory. | Pass |
| 3 | Looking at the count value when entering inputs | IDLE | The count value should be the same if the inputs are the same. | The count values are the same if the inputs are the same, the counts would be 1 or higher only if the inputs are different. | Pass |
| 4 | The conversion from input to binary is correct | IDLE, Browser | The output in binary should be able to convert to the same word that we input at the start | After converting the binary output from the script, we got the same input as the one we initially entered. | Pass |

## Test 1

Here I ran the script using IDLE and used words, numbers, and random characters as my inputs.

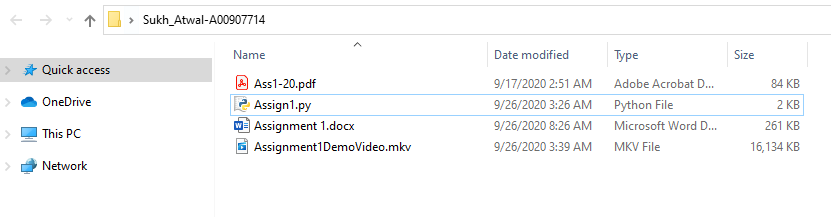




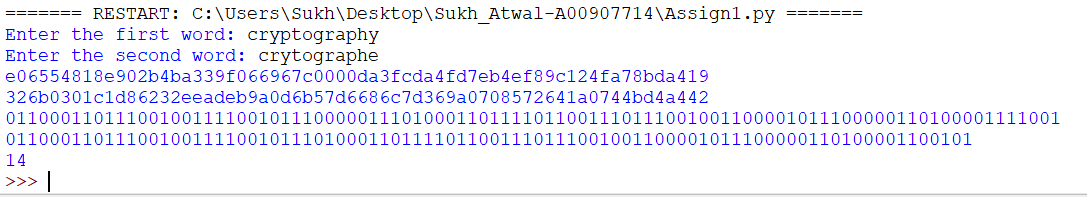
You can see that the hash and binary values were outputted correctly for each attempt.

## Test 2

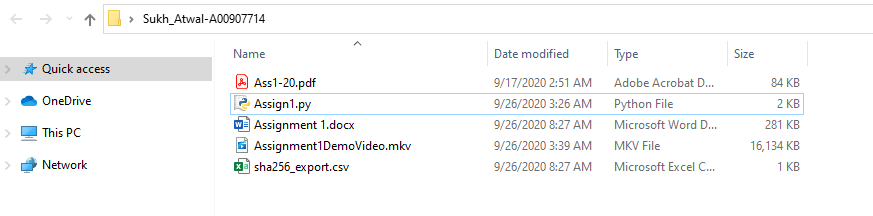
Here you can see that I have the python script in the following folder.



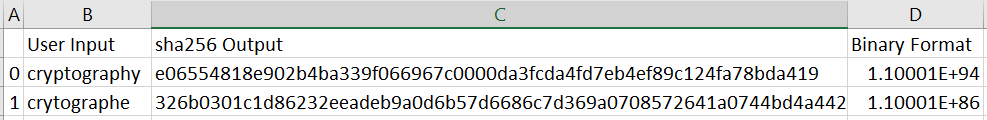
Now I’ll run the script and enter the following inputs.



As you can see, I get the hash and binary values in the console.



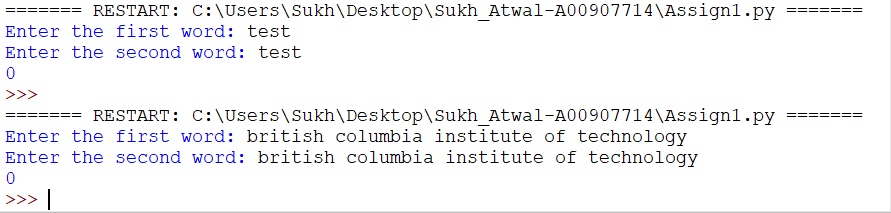
There is also a sha256\_export.csv file now in the directory.



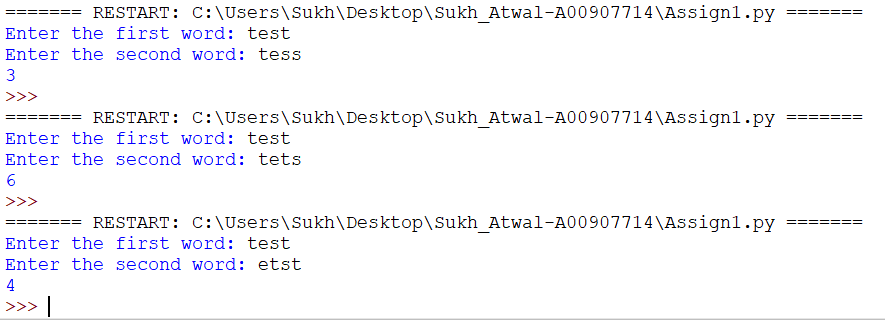
Here I open it and get both inputs, the hash value as well as the binary value as well which are the same as the one displayed in the console.

## Test 3

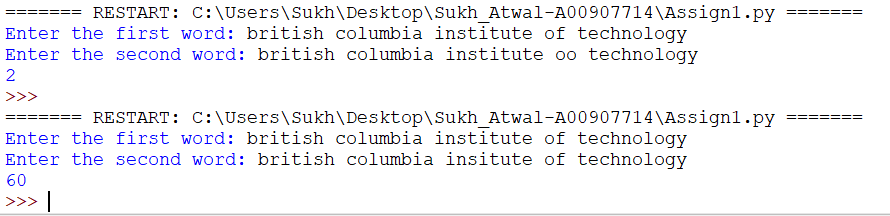
For the purposes of this test, I commented out the hash and binary values so that the only output in the console would be the difference in bit counts.



Above we can see that when the inputs are the same, there is a count of 0, meaning there is no difference in bits between the two.



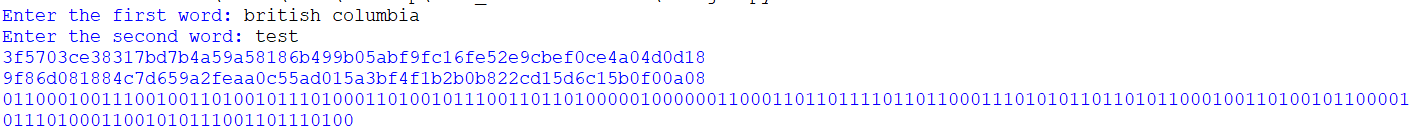
When we have different inputs, you can see the count value has gone up to 3, 6 and 4 respectively.



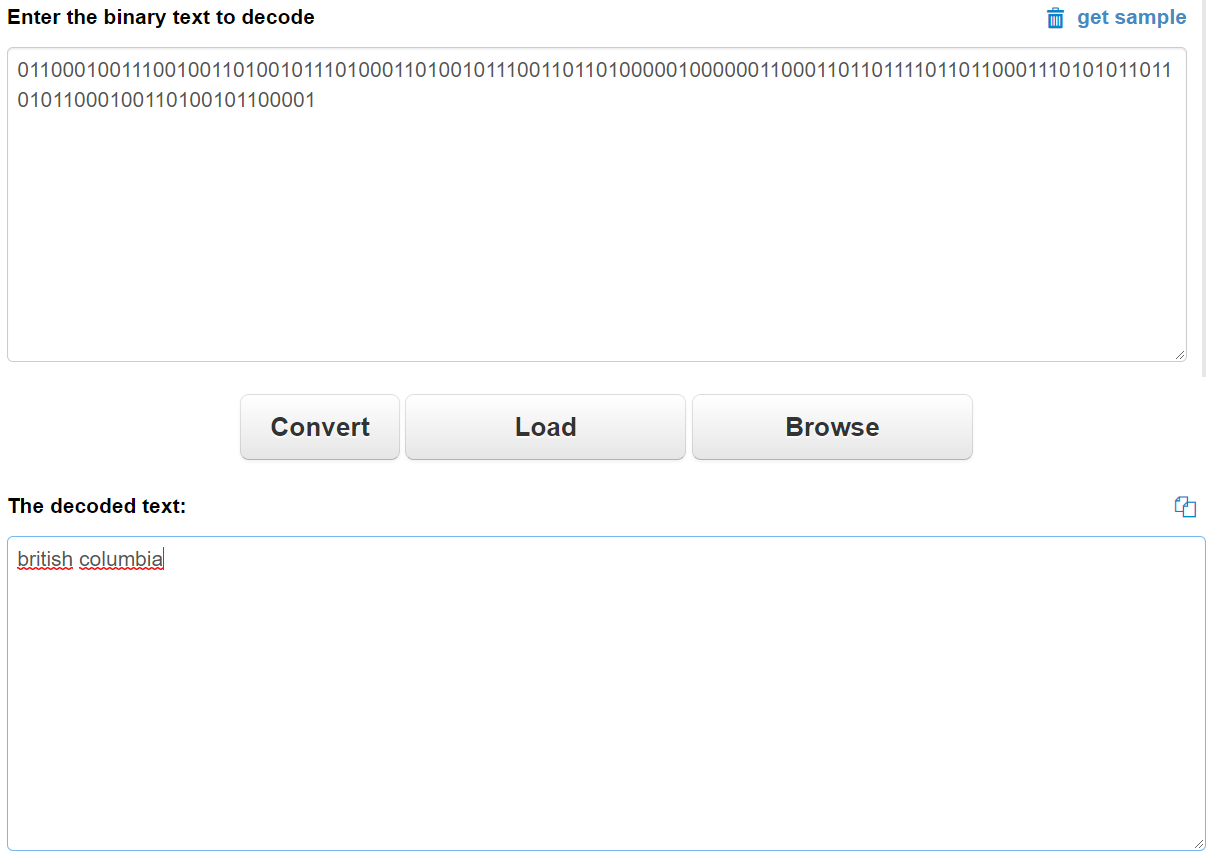
Using the second input example, we can see again the count value of bits changed is 2 and 60 for the following inputs.

Test 4

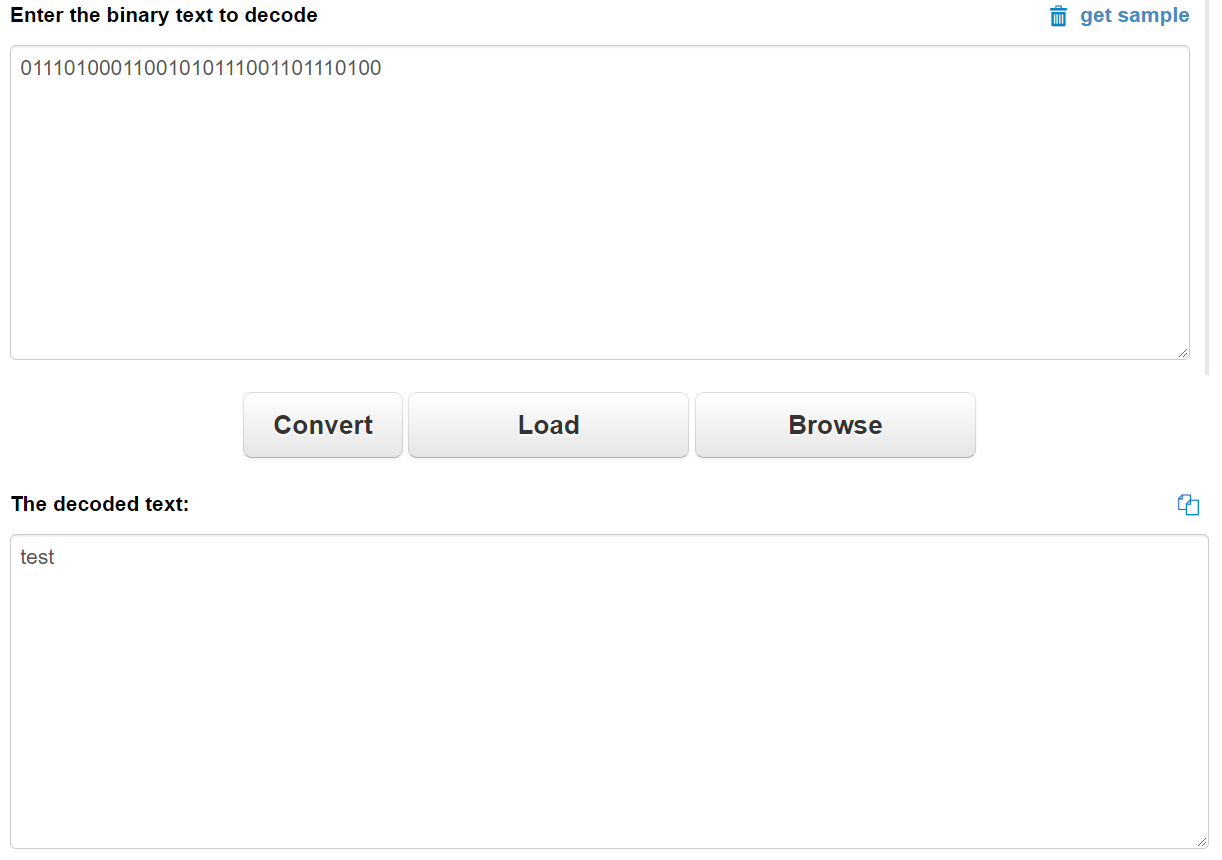
The two words I am using to test if the input to binary is working correctly are british columbia and test.



The website I am using is codebeautify for this scenario found at <https://codebeautify.org/binary-to-text>.



Starting off with british columbia which had a binary value of 01100010011100100110100101110100011010010111001101101000001000000110001101101111011011000111010101101101011000100110100101100001 in the script output. We can see it correctly converted it to the input ‘british columbia’.



Next, I looked at test which had a binary value of 01110100011001010111001101110100 in the script output. As we can see again, it correctly converted it to the input ‘test’.

# Bibliography

Kanumarlapudi, P. (2018, February 20). *BlockChain for layman — Part 2*. Retrieved from medium: https://medium.com/@pkmar437/blockchain-for-layman-part-2-a8984fda0acc