Lab 3
Let's Encrypt!

Student 1 – FULL NAME: Zachary D'Angelo Student 1 – Student/Banner ID: 100865977

Student 2 – FULL NAME: ______________ Student 2 – Student/Banner ID: ______________

Tools: `JCrypTool and OpenSSL`

Submission expectations: In this lab, you must write your answers in the provided regions on this lab manual, save, and submit the saved copy online (DO NOT CREATE A SEPARATE PDF DOCUMENT).

Download JCrypTool

In this exercise, you will download, extract and open JCrypTool – a graphical cryptography tool used for educational purposes.

1. Open the terminal and download JCrypTool using the following command:

```
wget https://github.com/jcryptool/core/releases/download/1.0.8/JCrypTool-1.0.8-Linux-64bit.tar.gz
```

2. Extract the downloaded file using the following command:

```
tar -xf JCrypTool-1.0.8-Linux-64bit.tar.gz
```

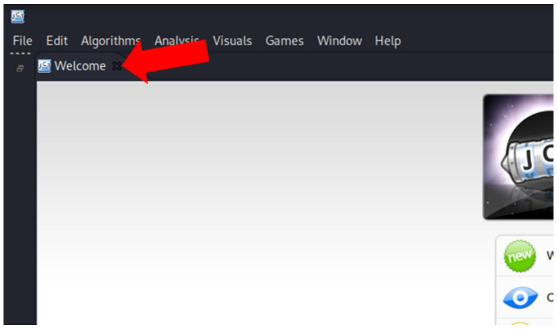
3. Change the directory to the extracted directory for JCrypTool using the following command:

```
cd jcryptool
```

4. Run JCrypTool using the following command:

```
./JCrypTool
```

5. Close the "Welcome" page as shown in the following figure:



1 Classic Cryptography [10 points]

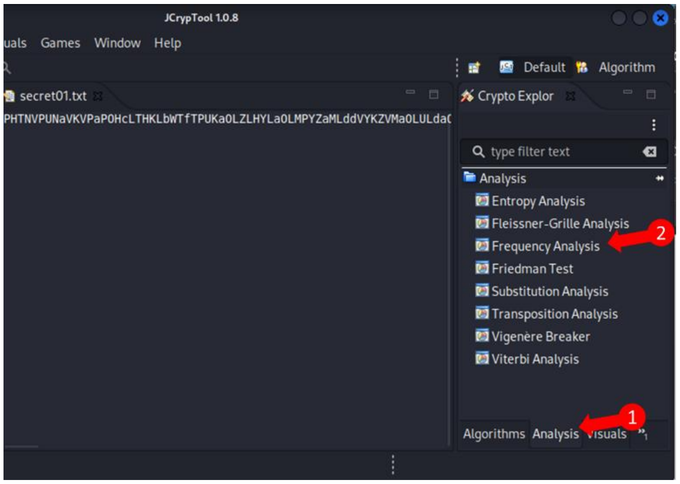
In this exercise, you will learn how to perform frequency analysis and decrypt a given cipher text created using Caesar cipher.

1.1 Frequency Analysis

1. Open the terminal and download “secret1.txt” using the following command:

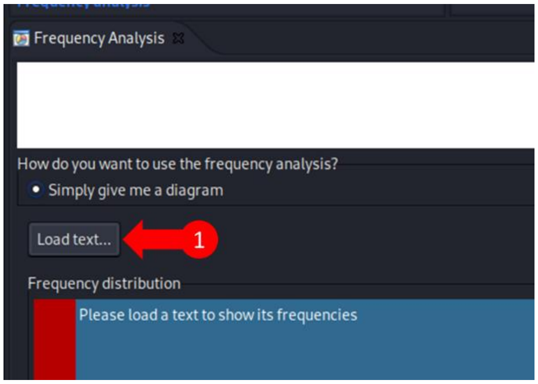
```
wget http://pmadani.ca/resources/secret1.txt
```

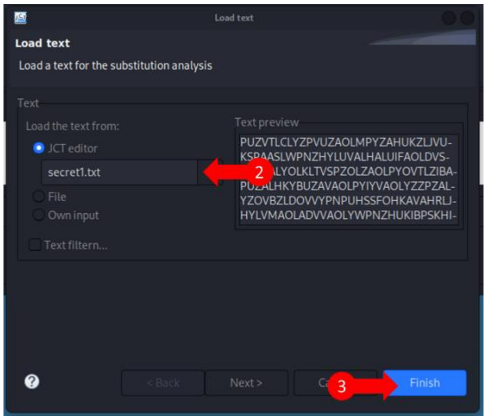
2. Open “secret1.txt” in JCrypTool from File>Open File>Home.
3. Can you comprehend the content of “secret1.txt”? No
4. Open “Frequency Analysis” as depicted in the following figure:



```
PHTNVPUNaVKVPaPOHcLTHKLbWTfTPUKaOLZLHYLaOLMPYZaMLddVYKZVMaOLULdaC
```

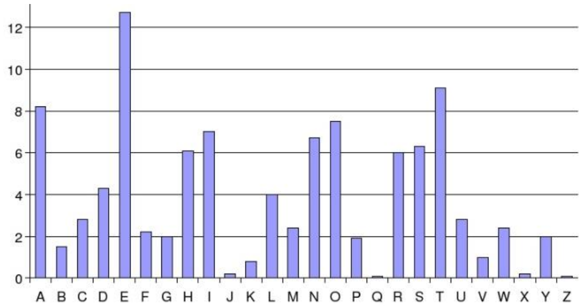
5. Load “secret1.txt” into “Frequency Analysis Module” by clicking on “Load text …”:


Frequency Analysis
How do you want to use the frequency analysis?
● Simply give me a diagram
Load text...
1
Frequency distribution
Please load a text to show its frequencies

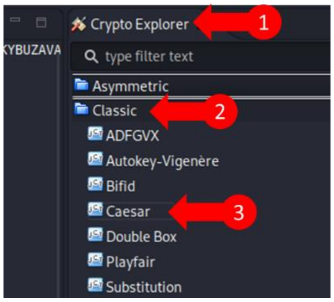


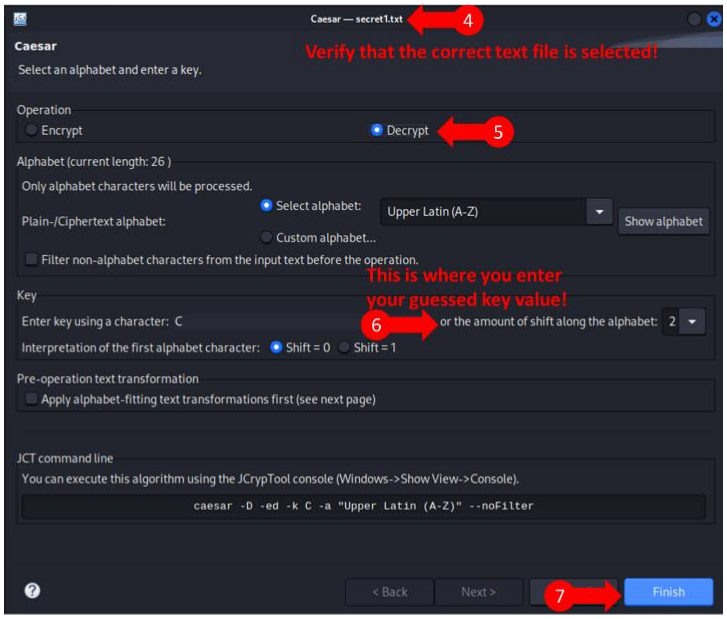
```
PUZVTLCLYZPVUZAOLMPYZAHUKZLJVU-
KSPAASLWPNZHYLUVALHALUIFAOLDVS-
ALYOLKLTVSPZOLZAOLPYOVTLZIBA-
PUZALHKYBUZAVAOLPYIYVAOLYZZPZAL-
YZOVBZLDOVVYPNPUHSSFOHKAVAHRLJ-
HYLVMAOLADVVAOLYWPNZHUKIBPSKHI-
```

6. Inspect the “Frequency Distribution” graph and compare it to the following “expected” frequency distribution of English letters:



7. What’s the encryption/decryption key ? 7
Try to guess the key used to produce this ciphertext using Caesar cipher by inspecting the two frequency distribution graphs.
Verify your guesses by using the “Caesar Cipher” decryption algorithm provided in JCrypTool:





Verify that the correct text file is selected!

Caesar

Select an alphabet and enter a key.

Operation
○ Encrypt ● Decrypt

Alphabet (current length: 26)
Only alphabet characters will be processed.
Plain-/Ciphertext alphabet: ● Select alphabet: Upper Latin (A-Z) Show alphabet
○ Custom alphabet...
□ Filter non-alphabet characters from the input text before the operation.

This is where you enter your guessed key value!

Key
Enter key using a character: C or the amount of shift along the alphabet: 2
Interpretation of the first alphabet character: ● Shift = 0 ○ Shift = 1

Pre-operation text transformation
□ Apply alphabet-fitting text transformations first (see next page)

JCT command line
You can execute this algorithm using the JCrypTool console (Windows->Show View->Console).

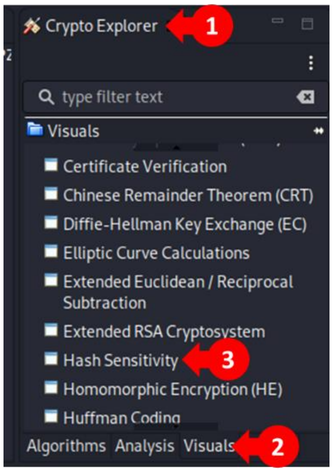
```
caesar -D -ed -k C -a "Upper Latin (A-Z)" --noFilter
```

8. Is the decrypted text comprehensible? What’s the first 10 characters: IN SOME VERS
Note: You must mentally add “spacing” to form words. If the decrypted text is still incomprehensible: (a) close the decrypted text window and (b) repeat step 7 using a new key value!

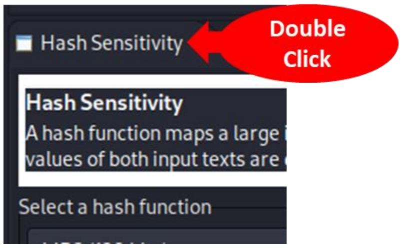
2 Hash Sensitivity [10 points]

In this exercise, you will observe the "avalanche effect", i.e., sensitivity of output values generated by cryptographic hash functions to slight changes made to the input data.

1. Open "Hash Sensitivity" from "Crypto Explorer>Visuals":

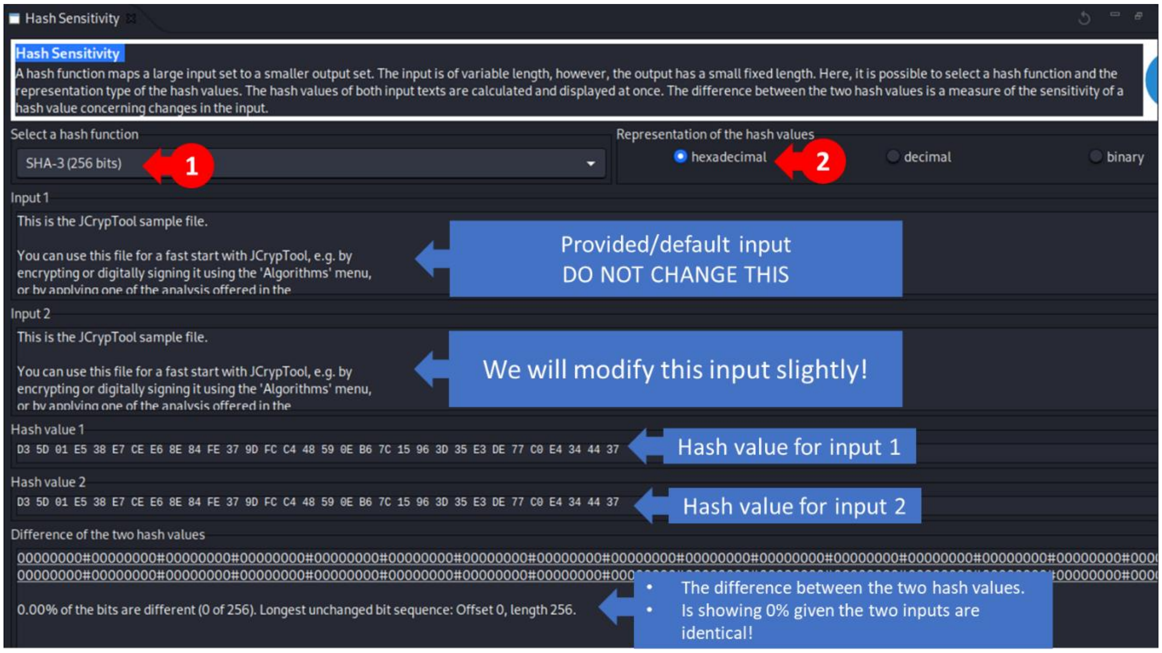

Crypto Explorer
1
type filter text
Visuals
Certificate Verification
Chinese Remainder Theorem (CRT)
Diffie-Hellman Key Exchange (EC)
Elliptic Curve Calculations
Extended Euclidean / Reciprocal Subtraction
Extended RSA Cryptosystem
Hash Sensitivity
3
Homomorphic Encryption (HE)
Huffman Coding
Algorithms
Analysis
Visuals
2

2. Double Click on the "Hash Sensitivity" Tab to maximize the size of the displayed tab:



Hash Sensitivity
A hash function maps a large
values of both input texts are

3. Set "Select a function" to "SHA-3 (256 bits)":



Hash Sensitivity
A hash function maps a large input set to a smaller output set. The input is of variable length, however, the output has a small fixed length. Here, it is possible to select a hash function and the representation type of the hash values. The hash values of both input texts are calculated and displayed at once. The difference between the two hash values is a measure of the sensitivity of a hash value concerning changes in the input.

Select a hash function: SHA-3 (256 bits)

Representation of the hash values: ● hexadecimal ○ decimal ○ binary

Input 1

This is the JCrypTool sample file.

You can use this file for a fast start with JCrypTool, e.g. by encrypting or digitally signing it using the 'Algorithms' menu, or by applying one of the analysis offered in the

Provided/default input
DO NOT CHANGE THIS

Input 2

This is the JCrypTool sample file.

You can use this file for a fast start with JCrypTool, e.g. by encrypting or digitally signing it using the 'Algorithms' menu, or by applying one of the analysis offered in the

We will modify this input slightly!

Hash value 1

```
D3 5D 01 E5 38 E7 CE E6 8E 84 FE 37 9D FC C4 48 59 0E B6 7C 15 96 3D 35 E3 DE 77 C0 E4 34 44 37
```

Hash value for input 1

Hash value 2

```
D3 5D 01 E5 38 E7 CE E6 8E 84 FE 37 9D FC C4 48 59 0E B6 7C 15 96 3D 35 E3 DE 77 C0 E4 34 44 37
```

Hash value for input 2

Difference of the two hash values

```
00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#0000
00000000#00000000#00000000#00000000#00000000#00000000#00000000#00000000#000                                                00000000#0000

0.00% of the bits are different (0 of 256). Longest unchanged bit sequence: Offset 0, length 256.
```

- The difference between the two hash values.
- Is showing 0% given the two inputs are identical!

4. The module is loaded with two (equal) default inputs, i.e., “Input1” and “Input 2”.
5. Change the first letter of Input 2 from “T” to “t” (making it lowercase!).
6. What’s the difference between the two hash values (in percentage)? 46.09%
   - This is known as the avalanche effect! Changing a single character in the input causes significant changes to the output value!

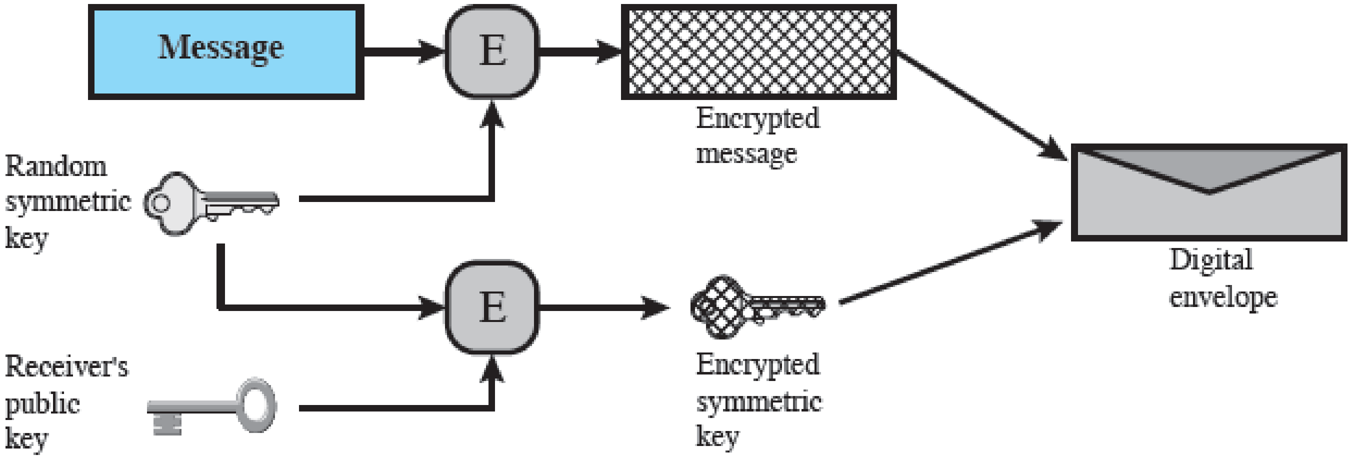
7. Change Input 2 as you like and as much as you want. What’s the highest “difference” percentage value you could get? 54.30%

8. As you can see, even when you changed Input 2 completely, there are still some bits in the output value that remained unchanged! Why is this an ideal behaviour for a secure hash function?

   This is an ideal hash function behaviour due to all three of their security criteria. This includes the preimage resistance, second preimage resistance, and strong collision resistance.

3 RSA Digital Envelope [20 points]

In class, we discussed the details of “RSA Digital Envelope” as depicted in the following figure:



In this exercise, you are going to implement this schema using OpenSSL.

- First, you begin by creating Alice’s public/private key pair.
- Second, Bob will encrypt his love letter using a symmetric key and AES block cipher.
- Third, Bob will encrypt the symmetric key using Alice’s public key.
- Finally, Bob will send the encrypted key and message to Alice!

3.1 Create Alice’s Public/Private key pair

1. Open a new terminal.
2. Create a public/private RSA key pair for Alice using the following command:

```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -pkeyopt rsa_keygen_pubexp:3 -out privkey-alice.pem
```

3. Inspect the generated private key using the following command:
   * Note: you should now be familiar with the content of the private key. All the numbers are represented in hexadecimal.

```
openssl pkey -in privkey-alice.pem -text
```

4. What is the public exponent? 3 (0x3)
5. What are the first ten characters of the private exponent? 00:99:9c:36:55
6. What are the first ten characters of the modulus (i.e., $\phi(n)$)? 00:e6:6a:51:80
7. Extract the generated public key into a separate file (that Alice would share with Bob!) using the following command:

```
openssl pkey -in privkey-alice.pem -out pubkey-alice.pem -pubout
```

Create the secret message and symmetric key

1. Create a text file containing the symmetric key that Bob will use to encrypt the secret that he will send to Alice:

```
echo “lab3” > symmetric_key.txt
```

2. Create a text file containing Bob’s secret message:

```
echo “Hi Alice, people say that love is in every corner … I must be walking in circles.” > message_plaintext.txt
```

3.2 Create the digital envelope

1. First, encrypt “message_plaintext.txt” using AES cipher with “lab3” as the encryption password:

```
openssl enc -aes-128-cbc -in message_plaintext.txt -out message_ciphertext.bin
```

   - Note: you can perform decryption using the following command – you need this knowledge for 3.4:

```
openssl enc -d  -aes-128-cbc -in name_of_the_file_encrypted_using_aes
```

2. Now, encrypt “symmetric_key.txt” (containing the decryption password) using Alice’s public key:

```
openssl pkeyutl -encrypt -in symmetric_key.txt -pubin -inkey pubkey-alice.pem -out encrypted_symmetrickey.bin
```

3. Finally, Bob will send “message_ciphertext.bin” and “encrypted_symmetrickey.bin” to Alice.
   - Since the encrypted_symmetrickey.bin is encrypted using Alice’s public key, she can decrypt the content and find out the symmetric encryption password used to generate “message_ciphertext.bin”:

```
openssl pkeyutl -decrypt -in encrypted_symmetrickey.bin -inkey privkey-alice.pem
```

3.3 Your turn!

Now, it is your turn to use my private-key information to decrypt the content of an RSA digital envelope that was created for me! Someone sent me a riddle, see if you read the message!

- Download `riddle-encrypted-key.bin` (containing the private key), `privkey-pooria.pem`, and `prof-madani-riddle.bin` using the following commands:

```
wget http://pmadani.ca/resources/riddle-encrypted-key.bin
wget http://pmadani.ca/resources/privkey-pooria.pem
wget http://pmadani.ca/resources/prof-madani-riddle.bin
```

- What’s the secret message? ____
  #elnaz-rekabi