

CSI4108 Assignment 3

Hongyi Lin, 300053082

- [CSI4108 Assignment 3](#)
 - [General](#)
 - [Question 1](#)
 - [Question 2 and Question 3](#)
 - [Question 4](#)
-

General

`math_helper.py` contains the helper functions to do the required math operations.

Question 1

source code: `q1.py`

```
TERMINAL  PROBLEMS  DEBUG CONSOLE  OUTPUT  JUPYTER
C:\Users\honyl\Desktop\CSI4108\A4 (a4)
> python .\q1.py
My result:
b'\xc2\xe9\x9f#\x88U\xe3\x1d\xbf\xbc\xca\xf3\r[\xdf\xda\xee\x13\x87\xad#d)fH$\x9e\x88\xf1_\x8f\xb9\xd2\x1d'P\x05\x82\xf7\t\x95\xc265\xba\xff\x15\x1b\xb9\x9f\xd6\x1b"\xea\xff\xbbuP\x8f\xbb\x8e'
Python hmac lib result:
b'\xc2\xe9\x9f#\x88U\xe3\x1d\xbf\xbc\xca\xf3\r[\xdf\xda\xee\x13\x87\xad#d)fH$\x9e\x88\xf1_\x8f\xb9\xd2\x1d'P\x05\x82\xf7\t\x95\xc265\xba\xff\x15\x1b\xb9\x9f\xd6\x1b"\xea\xff\xbbuP\x8f\xbb\x8e'
HMAC results match!
C:\Users\honyl\Desktop\CSI4108\A4 (a4)
> |
```

Question 2 and Question 3

source code:

- `dsa_num_gen.py`: Generate the system parameters of DSA, 1024-bit prime `p`, 160-bit prime `q`, and the generator `g`.
- `dsa_num.py`: Generated files by `dsa_num_gen.py` containing the DSA system parameters.
- `dsa.py`: Contain the `Dsa` class to implement the DSA algorithm with the sign and verify operations. This class use `hashlib.sha1` as the default hash function as required by the question 2.
- `q2_and_q3.py`: the script to run the simulation described in the PDF.

```

C:\Users\honyl\Desktop\CSI4108 (a4)
> & C:/Users/honyl/AppData/Local/Programs/Python/Python38/python.exe c:/Users/honyl/Desktop/CSI4108/a4/q2_and_q3.py
Question 2 signature is verified!

Question 3 signature is verified!

compromised k = 1055756312382330868406738829345578425239309471647
original k     = 1055756312382330868406738829345578425239309471647
k is compromised!

compromised x (private key) = 577063912867359432467087131777938929651186868698
original x (private key)    = 577063912867359432467087131777938929651186868698
x (private key) is recovered using compromised k!

C:\Users\honyl\Desktop\CSI4108 (a4)
> & C:/Users/honyl/AppData/Local/Programs/Python/Python38/python.exe c:/Users/honyl/Desktop/CSI4108/a4/q2_and_q3.py
Question 2 signature is verified!

Question 3 signature is verified!

compromised k = 867994914271609134440538131105383242985339017019
original k     = 867994914271609134440538131105383242985339017019
k is compromised!

compromised x (private key) = 11855378606032311784129123144855909403387586329
original x (private key)    = 11855378606032311784129123144855909403387586329
x (private key) is recovered using compromised k!

C:\Users\honyl\Desktop\CSI4108 (a4)
> & C:/Users/honyl/AppData/Local/Programs/Python/Python38/python.exe c:/Users/honyl/Desktop/CSI4108/a4/q2_and_q3.py
Question 2 signature is verified!

Question 3 signature is verified!

compromised k = 461367728452459072093447285632128271607963668266
original k     = 461367728452459072093447285632128271607963668266
k is compromised!

compromised x (private key) = 269403068927495656986215549634052058241918752881
original x (private key)    = 269403068927495656986215549634052058241918752881
x (private key) is recovered using compromised k!

C:\Users\honyl\Desktop\CSI4108 (a4)
> |

```

As we can see the results in the screenshot above, the **k** is compromised each time from the two signatures using the same **k**. From the compromised **k**, the private key **x** can also be recovered each time.

Question 4

Build the block cipher based on the Feistel Network, and use the hash functions instead of the S-boxes in the round function in Feistel Network.

Because of the nice property of the Feistel Network, the hash functions and round function are not required to be invertible. Each time the decryption is needed, just use the round keys in the reverse order to decrypt the ciphertext.