Week 8 Practical: Hashing & Elliptical Curve Cryptography (ECC) with *PyCrypto*

Dr. Qublai Ali Mirza

University of Gloucestershire

qalimirza@glos.ac.uk

Overview

1. Recap
2. Hashing
3. ECC
4. Bringing it all together
5. Post-sessional work

UNIVERSITY OF GLOUCESTERSHIRE

Recap

- Last week, we looked at using `PyCrypto` to implement:
  - RSA
  - DES
  - AES
- This week we will be looking at how to go about using hashing and Elliptical Curve Cryptography
- But before doing so, 10 minutes to complete the post-sessional work from last week.

MD5

- Designed by Ron Rivest in 1991
- Based on a non-linear function *F* which involves
  - Modular addition
  - Left rotation
- A flaw was identified in 1996
- Considered no longer collision resistant by 2004

Brief MD5 operation

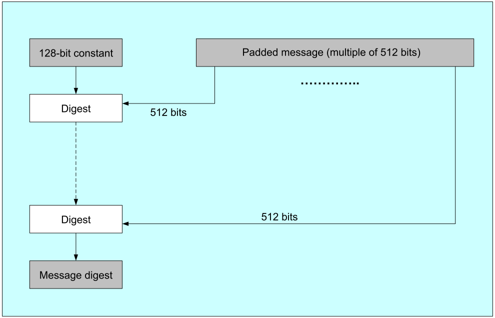

Figure: MD5 Operation recap

MD5 in Pycrypto I

```
import os

from Crypto.Hash import MD5

def calcFileHash(filename):

        # initialise a new MD5 object

        h = MD5.new()

        # Preset buffer size

        buffer_size = 8192
```

MD5 in Pycrypto II

```
with open(filename, 'rb') as f:

        while True:
                text = f.read(buffer_size)

                if len(text) == 0:

                        break

                h.update(text)

return h.hexdigest()
```

MD5 in Pycrypto III

```
if __name__ == '__main__':
    strHash = calcFileHash('plaintext.txt')

    print("Hash value: ", str(strHash))
```

Exercise

Based on your understanding of the lab materials from last week, develop a `Python` program that:

1. Reads the contents of `plaintext.txt`
2. Encrypts it using *AES*
3. Stores the output in a file called `ciphertext.txt`
4. Obtains the *MD5* hash of `ciphertext.txt`

SHA-256

- SHA originally designed by NIST & NSA in 1993
- Revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
- Based on design of MD4 with key differences
- Produces 160-bit hash values
- 2005 results on security of SHA-1 raised concerns on its use in future applications

SHA-256 I

```
import os

from Crypto.Hash import SHA256

def calcSHA256FileHash(filename):

    # initialise a new SHA256 object

    h = SHA256.new()

    # Preset buffer size

    buffer_size = 8192
```

SHA-256 II

```
with open(filename, 'rb') as f:

        while True:
                text = f.read(buffer_size)

                if len(text) == 0:

                        break

                h.update(text)

return h.hexdigest()
```

SHA-256 III

```
if __name__ == '__main__':
    strHash = calcSHA256FileHash('plaintext.txt')

    print("Hash value: ", str(strHash))
```

ECC I

```
import seccure

# Set the plaintext string

strPlainText = b'This is a test string for ECC encryption'

# Generate the public key based on a given passphrase

strPublicKey = str(seccure.passphrase_to_pubkey(b'Hello
                                        world'))

# Encrypt the plaintext using the public key
```

ECC II

```
strCipherText = seccure.encrypt(strPlainText,
                                strPublicKey)

print("Encrypted Text: ", str(strCipherText))

# Decrypt ciphertext using the passphrase

strDecryptedText = seccure.decrypt(strCipherText,
                                        b'Hello world')

print("Decrypted Text: ", str(strDecryptedText))
```

UNIVERSITY OF GLOUCESTERSHIRE

Bringing it all together

- We had a more in-depth look into `PyCrypto`
- We also looked at how we can implement hashing as well as ECC
- Next week: *Digital signatures & Certificates*

Post-sessional work

- Using the in-lab exercise at starting point, perform encryption on `plaintext.txt` using *any* encryption algorithm and measure the amount of time in *both* MD5 and SHA-256 hashing.
- **Hint:** you might want to use the `timeit.timeit` function

Q & A