Lab 12-08-2024

1. Do some research on the following signature algorithms
   - RSA
   - DSA

   then answer the questions at this link

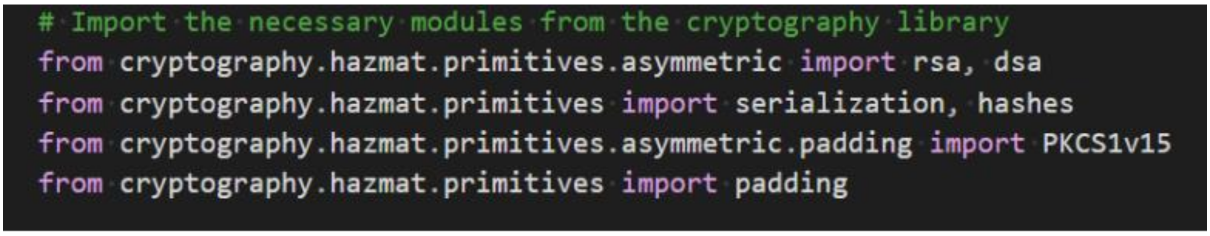
   https://forms.gle/j31wXtLNCB2MY2Sa7

2. Write a program that does the following
   a) Takes a plain text message
   b) Generates the public and private key pair for RSA
   c) Encrypts and decrypts the message using RSA
   d) Generates a public and private key pair for DSA
   e) Generates a signature for the message and verifies it using DSA

Instructions:

a) Note: For this hands on exercise you need to install the "cryptography" library. You can install it by "**pip install cryptography**".

b) To start, you need to import necessary modules from the cryptography library.



```
# Import the necessary modules from the cryptography library
from cryptography.hazmat.primitives.asymmetric import rsa, dsa
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric.padding import PKCS1v15
from cryptography.hazmat.primitives import padding
```

3. For the rest of the exercise please consult the following pseudo code.

   Function generateRSAKeyPair():

   1. Generate a private key using rsa.generate_private_key() function with public_exponent and key_size as parameters. Save the result in privateKey.
      a. Set public_exponent to 65537
      b. Set key_size to 2048
   2. Obtain the corresponding public key by calling privateKey.public_key(). Save the result in publicKey.
   3. Return privateKey and publicKey.

   Function RSAEncrypt(publicKey, plainText):

   1. Encrypt plainText using the encrypt(plainText, padding technique) method of publicKey. Use PKCS1v15() padding. Save the result in cipherText.
   2. Return the resulting cipherText.

   Function RSADecrypt(privateKey, cipherText):

   1. Decrypt cipherText using the decrypt() method of privateKey. Use PKCS1v15() padding.
   2. Return the resulting plainText.

   Function generateDSAKeyPair():

   1. Generate a private key using dsa.generate_private_key() function with key_size as a parameter. Save the result in privateKey.
      a. Set key_size to 1024
   2. Obtain the corresponding public key by calling privateKey.public_key(). Save the result in publicKey.
   3. Return privateKey and publicKey.

Function DSASign(privateKey, message):

1. Generate a signature by calling the sign(message, hashing algorithm) method of privateKey. Use hashes.SHA256() as the hashing algorithm.
2. Return the resulting signature.

Function DSAVerify(publicKey, message, signature):

1. Try to verify the signature by calling the verify(signature, message, hashing algorithm) method of publicKey. Use hashes.SHA256() as the hashing algorithm.
2. If the verification succeeds, return True.
3. If an exception occurs (doesn’t verify) during verification, return False.

Function main():

1. Generate an RSA key pair by calling generateRSAKeyPair(). Store the returned keys as RSAprivateKey and RSApublicKey.
2. Convert the message e.g "Message for RSA algorithm" to bytes and store it in plainText.
3. Encrypt plainText using RSAEncrypt(RSAPublicKey, plainText). Store the result in cipherText.
4. Decrypt cipherText using RSADecrypt(RSAPrivateKey, cipherText) function with privateKey. Store the result in decryptedText.
5. Print RSA details:
    a. RSA Public Key: RSAPublicKey
    b. RSA Private Key: RSAPrivateKey
    c. Plaintext: Decode plainText
    d. Ciphertext: cipherText
    e. Decrypted Text: decryptedText
6. Generate a DSA key pair by calling generateDSAKeyPair(). Store the returned keys as DSAPrivateKey and DSAPublicKey.

   7. Convert the message e.g "Message for DSA algorithm" to bytes and store it in a message.
   8. Generate a signature by calling the DSASign(DSAPrivateKey, message) function with privateKey and message as parameters. Store the result in signature.
   9. Verify the signature by calling DSAVerify(DSAPublicKey, message, signature) function with publicKey, message, and signature as parameters. Store the result in verified.
      a. Print DSA details:
      b. DSA Public Key: DSAPublicKey
      c. DSA Private Key: DSAPrivateKey
      d. Message: Decode message
      e. Signature: signature
      f. Verification: verified
   10. Call the main() function.

4. Write a program in the similar manner for elliptic curve digital signature algorithm (ECDSA). For your ease you can refer to the following pseudo code

   Function generateECDSAKeyPair():

   1) Generate a private key using ec.generate_private_key() function with ec.SECP256K1() as the parameter. Save the result in privateKey.
   2) Obtain the corresponding public key by calling privateKey.public_key(). Save the result in publicKey.
   3) Return privateKey and publicKey.

   Function ECDSASign(privateKey, message):

   1) Generate a signature by calling privateKey.sign() function with message and ec.ECDSA(hashes.SHA256()) as parameters. Save the result in signature.
   2) Return signature.

Function ECDSAVerify(publicKey, message, signature):

1) Try the following block of code:
    - a) Call publicKey.verify() function with signature, message, and ec.ECDSA(hashes.SHA256()) as parameters.
    - b) If the verification is successful, return True.
2) If any exception occurs (doesn’t verify), return False.

Function main():

1. Call generateECDSAKeyPair() function. Save the private key in ECDSAPrivateKey and the public key in ECDSAPublicKey.
2. Convert the message e.g. "Message for ECDSA algorithm" to bytes and store it in a message.
3. Call ECDSASign() function with ECDSAPrivateKey and message as parameters. Save the result in signature.
4. Call ECDSAVerify() function with ECDSAPublicKey, message, and signature as parameters. Save the result in verified.
5. Print "ECDSA:"
6. Print "ECDSA Public Key:" followed by the value of ECDSAPublicKey.
7. Print "ECDSA Private Key:" followed by the value of ECDSAPrivateKey.
8. Print "Message:" followed by the decoded value of message.
9. Print "Signature:" followed by the value of signature.
10. Print "Verification:" followed by the value of verified.

11. Call the main() function.