Lab 15-08-2024

1. In this lab you'll simulate signing a bitcoin transaction using Elliptic Curve Digital Signature Algorithm (ECDSA). For this purpose, please follow the following steps:
2. Import these necessary libraries at the top of your code

```
import hashlib
import random
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec
```

   [Note: For generation of random integers you can use the randint(min, max) function from random library in the following manner "random.randint(1, 1000000)"]
3. For the remaining steps please follow the following pseudocode

   Function generateTxid():

   Return the hexadecimal SHA-256 hash of a randomly generated integer converted to a string

   Function generateInput():

   a. prevTxid = generateTxid()
   b. prevOutputIndex = randomly generate an integer between 0 and 5
   c. Return prevTxid, prevOutputIndex

   Function generateOutput():

   a. recipientAddress = 'recipient_address_' + randomly generate an integer between 1 and 100 converted to a string
   b. amount = round a random floating-point number between 0.001 and 1.0 to 8 decimal places
   c. Return recipientAddress, amount

   Function generateTransactionFee():

   a. Return round a random floating-point number between 0.0001 and 0.001 to 8 decimal places

   Function generateRandomTransaction():

   a. txid = generateTxid()

b. inputPrevTxid, inputPrevOutputIndex = generateInput()
c. outputRecipientAddress, outputAmount = generateOutput()
d. transactionFee = generateTransactionFee()
e. Return txid, inputPrevTxid, inputPrevOutputIndex, outputRecipientAddress, outputAmount, transactionFee

Function concatenateString(txid, inputPrevTxid, inputPrevOutputIndex, outputRecipientAddress, outputAmount, transactionFee):

a. transactionData = concatenate the values of txid, inputPrevTxid, inputPrevOutputIndex, outputRecipientAddress, outputAmount, and transactionFee into a single string.
b. Convert non-string values to string data type by using str() function.
c. Return transactionData

Function generateECDSAKeyPair():

a. ECDSAPrivateKey = generate a private key using ec.generate_private_key(ec.SECP256K1())
b. ECDSAPublicKey = get the corresponding public key from ECDSAPrivateKey
c. Return ECDSAPrivateKey, ECDSAPublicKey

Function ECDSASign(privateKey, message):

a. signature = sign the message using privateKey.sign(message, ec.ECDSA(hashes.SHA256()))
b. Return signature

Function ECDSAVerify(publicKey, message, signature):

a. Try:
   Verify the signature using publicKey.verify(signature, message, ec.ECDSA(hashes.SHA256()))
   Return True
b. Catch any exception:
   Return False

Function main():

a. txid, inputPrevTxid, inputPrevOutputIndex, outputRecipientAddress, outputAmount, transactionFee = generateRandomTransaction()
b. transactionDataAsMessage = concatenateString(txid, inputPrevTxid, inputPrevOutputIndex, outputRecipientAddress, outputAmount, transactionFee).encode()
c. transactionDataAsMessageSHA256Hashed = calculate the SHA-256 hash of transactionDataAsMessage
d. ECDSAPrivateKey, ECDSAPublicKey = generateECDSAKeyPair()
e. signature = ECDSASign(ECDSAPrivateKey, transactionDataAsMessageSHA256Hashed)
f. verified = ECDSAVerify(ECDSAPublicKey, transactionDataAsMessageSHA256Hashed, signature)
g. Print "ECDSA:"
   Print "ECDSA Public Key:", ECDSAPublicKey in a simple format
   Print "ECDSA Private Key:", ECDSAPrivateKey in a simple format
   Print "transactionDataAsMessageSHA256Hashed:", transactionDataAsMessageSHA256Hashed in hexadecimal format
   Print "Signature:", signature in hexadecimal format
   Print "Verification:", verified
h. Call main() function