**1. feladat: Kulcskezelés**

Hozz létre egy ITKoin\_01\_<*neved*>.py állományt az alábbi kiinduló tartalommal:

from Crypto.PublicKey import RSA

import json

from base64 import b64encode, b64decode

from pprint import pprint

class ITKoin:

 def \_\_init\_\_ (self):

 self.pending\_transactions = []

 self.unspent\_transactions = []

 self.sender\_inputs = []

 self.chain = []

 @staticmethod

 def generate\_rsa\_key(filename):

 rsakey = # generálj 2048 bites RSA kulcsot

 rsapublickey = # a kulcs publikus része kerüljön ide

 # print(rsakey)

 # pprint(rsakey)

 # print(vars(rsakey))

 # pprint(vars(rsakey))

 pprint(vars(rsakey))

 pprint(vars(rsapublickey))

 PEMrsakey = # PEM formátumra alakítsd az RSA kulcsot

 pprint(PEMrsakey)

 PEMrsapublickey = # PEM formátumra alakítsd a kulcs publikus részét

 pprint(PEMrsapublickey)

 privatekeyfilename = filename + 'priv.pem'

 f = open(privatekeyfilename, 'wb')

 f.write(PEMrsakey)

 f.close()

 publickeyfilename = filename + 'pub.pem'

 f = open(publickeyfilename, 'wb')

 f.write(PEMrsapublickey)

 f.close()

 return

A PyCryptodome dokumentáció alapján tanulmányozd az RSA kulcs létrehozásának, tárolásának, beolvasásának lehetőségeit:

https://pycryptodome.readthedocs.io/en/latest/src/api.html

-> package: Crypto.Publickey

-> RSA keys

Hozz létre 2048 bites RSA kulcsot és tárold el file-ban a teljes kulcsot, illetve a publikus részét PEM formátumban.

Tájékozódás az RSA kulcsok tárolási formátumairól:

https://tls.mbed.org/kb/cryptography/asn1-key-structures-in-der-and-pem

Töltsd be a teljes kulcsot, illetve a publikus kulcsot file-ból az alábbi eljárások elkészítésével:

 def load\_key (self, filename):

 privatekeyfilename = filename+'priv.pem'

 privatekeyfileobject = open(privatekeyfilename, 'r')

 privatekeyfilecontent = privatekeyfileobject.read()

 pprint(privatekeyfilecontent)

 rsakey = # olvasd be a kulcsot

 self.rsakey = rsakey

 pprint(vars(self.rsakey))

 # rsapublickey = # a kulcs publikus része kerüljön ide

 # self.rsapublickey = rsapublickey

 # pprint(vars(self.rsapublickey))

 return

 def load\_public\_key (self, filename):

 publickeyfilename = filename+'pub.pem'

 publickeyfileobject = open(publickeyfilename, 'r')

 publickeyfilecontent = publickeyfileobject.read()

 pprint(publickeyfilecontent)

 rsakey = # olvasd be a kulcsot

 rsapublickey = # a kulcs publikus része kerüljön ide

 self.rsapublickey = rsapublickey

 pprint(vars(self.rsapublickey))

 return

Állíts be jelszó alapú titkosítást az RSA kulcs tárolásakor. Hasonlítsd össze a PEM formátumú titkosítatlan és titkosított kulcsokat.

Jegyzőkönyvben szerepeljen:

- hiányzó függvények, kódrészletek működésének rövid leírása

- a kulcsodat létrehozó és eltároló konkrét függvényhívások

- RSA kulcsok (privát / publikus) tartalma

- RSA kulcsok tárolási formátumai, azok jellemzői

- publikus kulcsod

- titkosított tárolás melyik kulcsra és hogyan valósítható meg

**2. feladat: Lenyomatolás, aláírás**

A PyCryptodome dokumentáció alapján tanulmányozd a lenyomat (hash) készítés, RSA aláírás, valamint aláírásellenőrzés modulok függvényeit.

Készíts lenyomatoló objektumot tetszőleges adatból létrehozó eljárást az alábbiak szerint.

 @staticmethod

 def create\_hashobject(data):

 stringdump = json.dumps(data) # ez nem teljesen korrekt megoldás, de így egyszerű mindent byte stringgé konvertálni

 binarydump = stringdump.encode()

 hashobject = # hozz létre egy hash objektumot

 # töltsd be az objektumba a lenyomatolni kívánt byte stringet

 hashhexvalue = # számítsd ki a lenyomatot, hexa kódolással

 print(hashhexvalue)

 return hashobject

Készíts aláíró és ellenőrző eljárást az alábbiak szerint.

 def create\_signature(self, data):

 signatureobject = # hozz létre egy signature objektumot

 hashobject = # az adatot töltsd be egy hash objektumba a create\_hashobject(data) használatával

 signaturevalue = # készítsd el az aláírás értéket a sign függvénnyel

 print(signaturevalue)

 b64signaturevalue = # kódold base64 kódolással

 print(b64signaturevalue)

 print(b64signaturevalue.decode())

 return b64signaturevalue.decode()

 def verify\_signature(self, data, b64signaturevalue, rsapublickey):

 verifyobject = # hozz létre egy verify objektumot

 hashobject = # az adatot töltsd be egy hash objektumba a create\_hashobject(data) használatával

 signaturevalue = # dekódold base64 kódolással az aláírás értéket

 signatureerror = # ellenőrizd az aláírást

 validsignature = # értéke: True, ha az aláírás érvényes

 return validsignature

Jegyzőkönyvben szerepeljen:

- hiányzó függvények, kódrészletek működésének rövid leírása

- aláírást létrehozó és ellenőrző konkrét függvényhívások

- miért szükséges a json.dumps alkalmazása, mi a probléma vele (5 mondat)

- mi a lenyomatolás szerepe az aláírás és ellenőrzés során (5 mondat)

**3. feladat: Bányászás**

Készíts bányász eljárást új blokk létrehozására az alábbiak szerint:

 def mine(self):

 transactions\_to\_block = []

 # itt be kellene tölteni file-ból a pending\_transctions-t

 while len(self.pending\_transactions) != 0:

 tr = self.pending\_transactions.pop()

 # itt validálni kellene az adott tranzakciót

 transactions\_to\_block.append(tr)

 if len(self.chain) == 0:

 previous\_block\_hash = 1

 else:

 previous\_block = self.chain[-1]

 hashobject = # az előző blokkot töltsd be egy hash objektumba a create\_hashobject(data) használatával

 previous\_block\_hash = # számold ki az előző blokk lenyomatát hexa értékkel

 nonce = 0

# hozz létre egy új blokkot (block) az alábbi tartalommal: {'index': <blokk sorszáma nullától kezdve>, 'transactions': <tranzakció lista>, 'nonce': <futóindex>, 'previous\_block\_hash': <előző blokk lenyomata>}

 while True:

 hashobject = # az adott blokkot töltsd be egy hash objektumba a create\_hashobject(data) használatával

 block\_hash = # számold ki az adott blokk lenyomatát hexa értékkel

 if # ha a lenyomat első két byte-ja hexa 00, kilép a ciklusból

 break

 block['nonce'] += 1

 self.chain.append(block)

# itt ki kellene file-ba menteni a blokkláncot és kiüríteni a pending\_transactions állományt

 pprint(block)

 return

Jegyzőkönyvben szerepeljen:

- hiányzó függvények, kódrészletek működésének rövid leírása (ismétlések nélkül)

- bányászást indító konkrét függvényhívás

- milyen ellenőrzéseket kellene elvégezni a blokkba helyezni kívánt tranzakciókon (minimum 3 ellenőrzés, egy-egy mondatban)

- ha a blokk lenyomatban elvárt 0-ák számát eggyel megnövelem, mennyivel nő a proof of work nehézsége (indoklással)

A jegyzőkönyv pdf formátumú, jegyzokonyv\_-01\_<neved>.pdf nevű legyen.

Küldd el a kész ITKoin\_01\_<neved>.py állományt is.