Pràctica 3 - Criptografia

Presentació

En aquesta pràctica estudiarem com funcionen els bitcoins, una moneda digital que últimament s'ha fet molt popular en els mitjans de comunicació pel creixement vertiginós del seu valor al canvi en dòlars. L'objectiu de la pràctica és implementar-ne una variant molt simplificada. Tot i la simplificació, la variant que descriurem a continuació conserva en essència les bases criptogràfiques que donen seguretat als bitcoins, de manera que la comprensió del sistema simplificat us permetrà entendre com funciona el sistema real dels bitcoins de forma genèrica i també us pot ser de molt ajut si decidiu indagar una mica més el funcionament real dels bitcoins.

Descripció de la PAC/pràctica a realitzar

El bitcoin és un sistema de moneda digital basat en criptografia de clau pública i funcions hash. Concretament, utilitza la criptografia de clau pública per a realitzar signatures digitals, i les funcions hash per a garantir la integritat de la informació.

Un primer punt a tenir en compte per entendre aquest nou sistema és que un bitcoin, com a moneda, no és cap objecte digital. Els bitcoins es poden definir com a apunts comptables en un compte corrent. Cada compte corrent, identificat per una adreça, tindrà un seguit d'apunts comptables que són els que determinaran l'import, en bitcoins, associat a aquest compte corrent. Anomenarem a aquestes comptes corrents, comptes de bitcoins. Una particularitat important d'aquests apunts comptables és que no es poden fraccionar, és a dir, un apunt comptable d'entrada ha de coincidir en la seva totalitat amb un apunt comptable de sortida.

Un compte de bitcoins està identificat per una adreça. Aquesta adreça és el resultat d'aplicar una funció específica (i pública) a una clau pública d'un criptosistema que permet realitzar signatures digitals. D'aquesta manera, i simplificant, podríem dir que l'adreça d'un compte de bitcoins correspon a una clau pública. Més endavant veurem que només podrem gastar els imports d'un compte de bitcoins si tenim coneixement de la clau privada associada a la clau pública de la seva adreça.

Així doncs, donat que un compte de bitcoins és simplement un identificador que està associat a un parell de claus pública-privada, és fàcil d'entendre que qualsevol usuari pot generar el seu compte de bitcoins simplement generant el parell de claus pública-privada. La clau pública determinarà l'adreça del seu compte corrent de bitcoins i la clau privada, que haurà de guardar-se en secret, li permetrà fer pagaments d'aquell compte de bitcoins.

Un cop sabem com es poden generar comptes de bitcoins, ens cal ara saber com es poden fer pagaments, és a dir, fer transferències de bitcoins d'un compte de bitcoins a un altre. Per fer aquests pagaments, el sistema de bitcoins defineix el que s'anomenen transaccions. Una transacció no és res més que una transferència de bitcoins entre dos comptes. Veiem amb un exemple com funciona:

Suposem que l'usuari A genera un parell de claus pública-privada, que denominarem $\{PK_A, SK_A\}$. Si la funció f és la que genera l'adreça a partir de la clau pública, tenim que l'adreça del compte de bitcoins de l'usuari A serà $Adre_A = f(PK_A)$. De forma equivalent, suposem que l'usuari B té les claus $\{PK_B, SK_B\}$ i l'adreça $Adre_B = f(PK_B)$ [1]. A més, suposarem també que al compte amb adreça $Adre_A$ hi ha 25 bitcoin, que denotarem per BTC. En aquesta situació, per tal que A pogués transferir els 25 BTC a B el que hauria de fer és crear una transacció amb la següent informació:

$$\{Adre_A, PK_A, 25\,BTC, Adre_B, Sig_{SK_A}(Adre_A, 25\,BTC, Adre_B)\}$$

Fixeu-vos que en la transacció s'hi indica que la primera adreça que hi apareix (adreça origen) fa una transferència de 25 BTC a la segona adreça que hi apareix (adreça destí) i per verificar que la transacció és correcta, qui fa la transacció, l'usuari A, realitza una signatura digital de la informació de la transacció utilitzant la seva clau privada SK_A.

L'usuari B, per tal de verificar que efectivament qui realitza la transferència és l'usuari A, realitza les següents verificacions. En primer lloc, calcula l'adreça $Adre_A$ a partir del valor PK_A utilitzant la funció f. Un cop validada que la clau pública correspon a l'adreça, podrà validar la signatura digital $Sig_{SK_A}(Adre_A, 25\,BTC, Adre_B)$ que l'usuari A ha realitzat, utilitzant justament la clau pública d'A, PK_A. Si la signatura digital es valida correctament, B acceptarà el pagament com a correcte.

Tal com hem comentat al principi, cal recordar que el sistema bitcoin només permet gastar una fracció de l'import d'un compte de bitcoins si aquesta fracció equival a la totalitat d'una transacció. Dit d'una altra manera, cada pagament d'un compte de bitcoins ha de provenir d'un cobrament d'exactament el mateix import realitzat en una transacció prèvia.

Arribats a aquest punt, i donat que estem estudiant una assignatura de seguretat, seria obvi interessar-se per la seguretat del sistema. I la primera pregunta que ens ve al cap sobre la seguretat d'aquest sistema és la següent:

Un cop l'usuari A ha realitzat la transacció:

$$\{Adre_A, PK_A, 25\,BTC, Adre_B, Sig_{SK_A}(Adre_A, 25\,BTC, Adre_B)\}$$

1 Fixeu-vos que cada usuari pot generar tants parells de claus pública-privada com vulgui i per tant pot tenir tants comptes com vulgui. A més, la identitat de l'usuari no té perquè estar relacionada amb els comptes en bitcoins que generi.

quin mecanisme hi ha perquè l'usuari A no pugui tornar a transferir els mateixos bitcoins? És a dir, què priva a l'usuari A de realitzar la transacció:

$$\{Adre_A, PK_A, 25\,BTC, Adre_C, Sig_{SK_A}(Adre_A, 25\,BTC, Adre_C)\}$$

amb la que estaria pagant 25 BTC a l'usuari C, i per tant, duplicant el valor dels seus bitcoins?

El mecanisme que utilitza el sistema bitcoin per resoldre aquest problema de seguretat (problema molt estudiat en sistemes de pagament electrònic i que rep el nom de sobredespesa) és basar-se en l'anotació i publicació de totes les transaccions que es realitzen en el sistema. Aquesta anotació es realitza per mitjà del que s'anomenen blocs. Per ara, en tenim prou en pensar en un bloc com un conjunt de transaccions, i més endavant, ja en definirem l'estructura exacta que, gràcies a l'explicació que anem fent del sistema, ens serà més comprensible. A més, tots els blocs que el sistema bitcoin va generant s'ajunten en el que s'anomena cadena de blocs. Només hi ha una única cadena de blocs en tot el sistema bitcoin i, per tant, aquesta cadena inclou totes les transaccions de tots els comptes de bitcoins que s'han realitzat. D'aquesta manera, i tornant a l'exemple anterior de la transferència entre els usuaris A i B, quan l'usuari B rep la transacció d'A, abans de realitzar les comprovacions que hem descrit, B ha de validar que l'adreça de bitcoins té l'import de 25 BTC. Per fer-ho, li caldrà analitzar la cadena de blocs. Per tal de facilitar la tasca, el que es fa és modificar l'estructura de la transacció que hem presentat en l'exemple anterior i deixar-la de la següent manera:

$$\{Adre_A, PK_A, HashPrevTrans, 25\,BTC, Adre_B,$$
$$Sig_{SK_A}(Adre_A, HashPrevTrans, 25\,BTC, Adre_B)\}$$

En el nou camp, $HashPrevTrans$, s'identifica la transacció anterior en la qual s'han transferit els bitcoins a $Adre_A$, és a dir, la transacció on $Adre_A$ figura com a adreça de destí[2]. Amb aquesta informació inclosa en les transaccions i tenint en compte que la cadena de blocs és de domini públic, el problema de la sobredespesa queda gairebé solucionat. Ara, donada una transacció, podem comprovar que aquesta no ha estat gastada amb anterioritat comprovant que no hi ha cap altra transacció a la cadena de blocs que contingui en el camp $HashPrevTrans$ el valor que ens apareix en el camp $HashPrevTrans$ de la nova transacció que volem validar.

Ara bé, donada la rigidesa que acabem de descriure, és a dir, la restricció que les transaccions s'han de fer per la totalitat del valor d'una transacció anterior, com podem realitzar transaccions per imports variables? Bitcoin soluciona aquest problema permetent que una mateixa transacció tingui vàries entrades, és a dir, vàries transaccions anteriors, $HashPrevTrans$, de les quals se'n gasta l'import, així com

2Recordem que hem indicat anteriorment que els pagaments, apunts comptables dels comptes de bitcoins, només es podien fer per la totalitat de l'import d'una transacció anterior, d'aquí que es pugui identificar una transacció com a "transacció anterior".

també vàries sortides, és a dir, varis destinataris entre els quals repartir l'import total. D'aquesta manera, l'estructura de la transacció que hem definit anteriorment es modifica per permetre aquesta flexibilitat:

$$Trans_{id} = \{[input_1, input_{2,} \ldots, input_n], [output_{1,} output_{2,} \ldots, ouput_m]\}$$

$$output_i = \{Adre_{Bi}, import_i\}$$

$$input_i = \{Adre_{Ai}, PK_{Ai}, HashPrevTrans, IndexOutputPrevTrans, Sig_{SKAi}(Adre_{Ai}, HashPrevTrans, IndexOutputPrevTrans, [output_{1,} \ldots, output_m])\}$$

Amb aquest esquema, una única transacció permet transferir una quantitat arbitrària de BTC a diferents comptes de bitcoin, que poden o no pertànyer a un mateix propietari.

A més, una transacció pot suposar la despesa de vàries transaccions anteriors indicant els valors hash de cadascuna d'aquestes transaccions. Noteu que cada una d'aquestes transaccions pot anar associada a una adreça diferent $Adre_{Ai}$. Per poder gastar les transaccions anteriors, caldrà demostrar que se n'és el propietari, creant la signatura amb la clau privada corresponent SK_{Ai} per a cada entrada. Com hem vist, les transaccions poden tenir diverses sortides i, per tant, quan especifiquem el hash de la transacció anterior que estem gastant, també caldrà especificar-ne la sortida concreta que volem gastar $IndexOutputPrevTrans$.

Per tal que la transacció sigui vàlida, la suma de tots els imports d'entrada ha de ser com a mínim la suma dels imports de sortida, és a dir, no podem gastar més BTC dels que hem demostrat que tenim.

Dèiem anteriorment que afegint el valor hash de la transacció que gastem a la nova transacció gairebé solucionava el problema de la sobredespesa. Aquest "gairebé" deixava entreveure que encara hi havia un últim punt a resoldre: la integritat de la cadena de blocs. És a dir, per tal que la verificació de no sobredespesa sigui correcta, cal assegurar que la cadena de blocs sigui única i que no pugui ser modificada.

Per obtenir la integritat de la cadena de blocs, l'objectiu és aconseguir que la inclusió dels blocs en la cadena sigui una tasca molt costosa i que a més cada bloc depengui de l'anterior. D'aquesta manera, modificar un bloc equival a incloure'n un altre amb dades diferents a la cadena de blocs (per tant una tasca difícil) i donat que els blocs estan encadenats, la dificultat de modificar una part de la cadena de blocs és proporcional al nombre de blocs que es volen modificar.

El mecanisme que utilitza el sistema de bitcoins per incloure un bloc en la cadena és realitzar el càlcul del valor hash del contingut del bloc. Òbviament, com hem vist en el mòdul de signatures digitals, el càlcul del hash d'un contingut digital és una operació molt ràpida de fer i no té cap mena de dificultat. La dificultat es troba en que per a la inclusió d'un bloc en la cadena cal que el càlcul del valor hash del bloc sigui més petit

que un cert valor donat, anomenat *target*. La possibilitat d'obtenir diferents valors hash per un mateix bloc (cosa que intuïtivament no és possible) passa per incloure un camp en cada bloc, anomenat *nonce*, el qual permet assignar-hi un valor aleatori. Per tant, el procés per aconseguir incloure un bloc en la cadena és un procés iteratiu en el qual es calcula el hash del bloc i es comprova si el valor resultant és inferior o igual al target donat. Si la comprovació és correcta, voldrà dir que ja hem pogut incloure el bloc en la cadena de blocs i si no ho és, modificarem el valor *nonce* del bloc i tornarem a calcular el hash. Fixeu-vos que aquest procés trobarà la solució amb més o menys dificultat depenent del valor del *target*. Suposem, per fer-ho fàcil, que la funció hash retorna un valor de 6 bits i assignem com a *target* el valor t = 111111. Donat que aquest valor del *target* és el més gran possible, incloure el bloc en la cadena és molt simple, ja que al calcular el primer hash del bloc, el valor que obtindrem serà segur un valor inferior o igual al *target*, perquè tots ho són! Ara bé, si fem el *target* més petit, per exemple, t = 001111, veiem que només ens serviran els hashos que tinguin com a sortida un valor que tingui dos zeros a l'inici. Donat que les funcions hash es consideren properes a generadors aleatoris (en tant que no es pot saber com modificar l'entrada per obtenir una modificació concreta de la sortida) la probabilitat de trobar aquest hash és de $2^4/2^6$ és a dir, haurem de generar una mitjana de 4 valors de hash per obtenir-ne un de més petit que el *target*. En el cas extrem, si el nostre *target* fos t = 00000, només ens servira el valor hash 000000, de manera que, en mitjana, hauríem de generar un total de 2^6 hashos per aconseguir incloure el bloc en la cadena.

Com es pot veure de la definició feta fins ara, el sistema dels bitcoins no és simple. A més, amb el que tenim definit fins aquí encara ens queda una pregunta per resoldre, intrínseca al funcionament del propi sistema: Com han pogut anar a parar els 25 BTC al compte $Adre_A$ del primer exemple? Amb la definició de transacció que hem donat, ens queda clar com es fa per passar BTC d'un compte a un altre, però això assumeix que els BTC ja estan en algun compte. Ara bé, com apareixen els bitcoins en un compte? O dit d'una altra manera, com es generen els bitcoins?

La generació dels bitcoins és un altre dels punts importants del sistema i el mecanisme per fer-ho és exactament el mateix que l'utilitzat per incloure un bloc en la cadena de blocs. De fet, s'utilitza el mateix mecanisme perquè el sistema està dissenyat per generar bitcoins nous cada vegada que s'inclou un bloc en la cadena. La generació de bitcoins es fa per mitjà d'una transacció especial que conté la següent informació:

$$Trans_{gen}\{Gen, 25\,BTC, Adre_A\}$$

Cada nou bloc que s'inclou a la cadena de blocs incorpora una única d'aquestes transaccions especials de generació, de manera que en cada inclusió de bloc en la cadena no només s'estan validant les transaccions que s'han fet fins al moment sinó que també s'estan creant nous bitcoins. Aquesta transacció indica que l'adreça $Adre_A$ ha generat 25 BTC. Així, el sistema recompensa, per mitjà dels nous bitcoins que es

creen, als usuaris que estan realitzant operacions per validar les transaccions[3]. El nombre de bitcoins que es crea en cada bloc depèn de la data de creació del bloc. El sistema està fet perquè cada vegada es creïn menys bitcoins. En el seu inici, cada bloc creava 50 BTC nous però en l'actualitat cada bloc ja només en crea 25.

Amb totes les explicacions que hem realitzat fins ara ja podem descriure el format que tindrà un bloc, un cop ja està inclòs en la cadena de blocs:

$$Bloc_i = \{hash(Bloc_{i-1}), nonce, target, Trans_{gen}, [Trans_{id_1}, \ldots, Trans_{id_n}]\}$$

on $hash(Bloc_{i-1})$ correspon al hash del bloc anterior de la cadena de blocs, $nonce$ conté el valor que fa que el hash del bloc sigui més petit que el target del moment, $target$ conté el valor de target actual (en el moment en que es genera el bloc), $Trans_{gen}$ conté la transacció de generació de bitcoins del bloc i $[Trans_{id_1}, \ldots, Trans_{id_n}]$ és una llista amb totes les transaccions que s'han inclòs en el bloc.

Simplificació del sistema

Un cop descrit el funcionament global del sistema de bitcoins i per tal que el sistema sigui molt més simple que el real i el pugueu implementar, assumirem les següents simplificacions:

- Les signatures digitals es realitzaran amb l'RSA.
- La funció hash, tant per realitzar signatures digitals com per incloure blocs a la cadena, serà l'MD5. Específicament, per obtenir el hash de qualsevol cadena farem servir la funció `UOC_MD5` que trobareu implementada en el fitxer sws de la pràctica.
- L'adreça d'un compte de bitcoins serà el valor MD5 de la seva clau pública. Específicament, l'adreça serà el resultat de la crida a la funció `UOC_MD5` amb la cadena formada per la concatenació del mòdul i l'exponent de la clau pública com a paràmetre.
- La funció `UOC_MD5` retorna una cadena amb la representació hexadecimal del hash. Per tal de signar aquesta cadena la convertirem a un enter fent servir la funció `hexString_to_int`.
- Cada transacció només es podrà fer entre una única adreça d'inici i una única adreça de destí.
- L'import de les transaccions es fixa en 25 BTC.

3 Tingueu en compte que una de les característiques rellevants del sistema de bitcoins és que no té cap autoritat central que controli el sistema ja que és un sistema P2P, de manera que qualsevol usuari pot validar transaccions i generar noves monedes a través de la inclusió de nous blocs a la cadena de blocs.

En l'arxiu sws que se us proporciona per a la implementació de la pràctica hi trobareu definida l'estructura d'un parell de claus RSA, d'una clau pública RSA, d'una transacció de bitcoins, d'un bloc i de la cadena de blocs. Aquestes definicions són les següents:

```
# RSA key pair structure
class rsa_key():
    def __init__(self):
        self.publicExponent = -1
        self.privateExponent = -1
        self.modulus = -1

    def __init__(self, publicExponent, privateExponent, modulus ):
        self.publicExponent = publicExponent
        self.privateExponent = privateExponent
        self.modulus = modulus

# RSA public key structure
class rsa_public_key():
    def __init__(self):
        self.exponent = -1
        self.modulus = -1

# Transaction structure
class transaction_struct():
    def __init__(self):
        self.transaction_hash = -1
        self.address_source = -1
        self.source_public_key_info = rsa_public_key()
        self.address_destination = -1
        self.tximport = -1
        self.hash_previous_transaction = -1
        self.signature = -0x01
```

```
# Block structure
class block_struct():
    def __init__(self):
        self.block_hash = -1
        self.previous_block_hash = -1
        self.target = -1
        self.bitcoin_gen_transaction = transaction_struct()
        self.transaction_list = []
        self.nonce = 0

# Blockchain structure
BLOCK_CHAIN = []
```

Totes les classes tenen un mètode `print_me()` que mostra el contingut de l'estructura. A més, tant les transaccions com els blocs tenen un mètode `get_hash_transaction()` (o `get_hash_block()`) que retorna una cadena amb tot el contingut de la transacció (o del bloc). Podeu fer servir aquesta cadena per calcular el hash de la transacció o del bloc.

Hi ha una única estructura per definir totes les transaccions (`transaction_struct`). Tant les transaccions de generació de bitcoins com les transaccions normals es representaran amb aquesta estructura. Definirem una transacció de generació de bitcoins com aquella que conté -1 al camp address_source.

Exercici 1 (5 punts)

En aquest exercici implementarem les funcions bàsiques de validació de la cadena de blocs.

1. Programeu una funció que validi una transacció de generació de bitcoins. **(0,5 punts)**

La funció validarà la correcció de la transacció. Per fer-ho, haurà de comprovar que la transacció està ben formada (per exemple, que el camp address_source està inicialitzat a -1 i que el hash és correcte) i que l'import de la transacció és l'import de generació del moment (en el nostre cas, 25 BTC).

a) `transaction`: estructura de la transacció del pagament.

La funció retornarà un "1" en cas que la transacció sigui vàlida i un "0" en cas contrari.

2. Programeu una funció que validi una transacció normal (no de generació de bitcoins)[4]. **(2 punts)**

La funció validarà la correcció d'una transacció. Per fer-ho, haurà de comprovar que la transacció està ben formada i que no hi ha cap transacció anterior que tingui el mateix hash_previous_transaction (no hi ha sobre despesa). A més, la funció haurà de validar la signatura digital de la transacció. Per fer-ho, no només haurà de validar la signatura utilitzant la clau pública proporcionada en la transacció, sinó que també caldrà que validi que aquesta clau pública correspon a l'adreça que figura en la transacció anterior. Per realitzar totes aquestes validacions, la funció rebrà com a argument la cadena de blocs i la transacció.

a) `block_chain`: cadena de blocs.

b) `transaction`: estructura de la transacció del pagament.

La funció retornarà un "1" en cas que la transacció sigui vàlida i un "0" en cas contrari.

Nota: Aquesta funció assumeix que la cadena de blocs que rep com a paràmetre és vàlida.

3. Programeu una funció que validi un bloc. **(1 punt)**

La funció validarà la correcció d'un bloc, és a dir, comprovarà que el hash del bloc és inferior al target indicat en el propi bloc i que cada una de les transaccions que s'hi inclouen és vàlida. També comprovarà que el bloc està ben format (el hash és correcte). Per fer-ho, la funció rebrà com a arguments el bloc i la cadena de blocs:

4 Fixeu-vos que la validació d'una transacció equival a la validació d'un pagament, acció que cal realitzar abans d'acceptar el pagament com a vàlid.

a) `block_chain`: cadena de blocs.

b) `block`: bloc a validar.

La funció retornarà un “1” en cas que el bloc sigui vàlid i un “0” en cas contrari.

Nota 1: Un bloc ha de tenir, com a mínim, una transacció de generació de bitcoins vàlida. No és necessari que un bloc contingui cap altra transacció.

Nota 2: Aquesta funció assumeix que la cadena de bitcoins que rep com a paràmetre és vàlida.

4. Programeu una funció que validi la cadena de blocs. **(1,5 punts)**

La funció validarà que cada un dels blocs de la cadena sigui correcte i que la cadena dels hashos dels blocs estigui ben contruida, és a dir que dins del bloc i-èssim hi ha el hash del bloc (i-1)-èssim. Per realitzar aquesta validació, la funció rebrà com a argument la cadena de blocs.

a) `block_chain`: l'estructura amb tota la cadena de blocs.

La funció retornarà, en cas que la validació de la cadena sigui correcta, el valor hash de l'últim bloc de la cadena i “0” en cas que la validació no sigui correcta.

Exercici 2 (4 punts)

En aquest exercici implementarem un moneder de bitcoins del nostre sistema simplificat. Per fer-ho, treballarem amb la classe `bitcoin_wallet`, que és l'encarregada de gestionar el moneder de bitcoins. L'estructura de la classe és la següent:

```
# Bitcoin Wallet
class bitcoin_wallet():
    def __init__(self):
        # Key storage: list of rsa keys
        self.keys = []


    # Returns the keys stored in the wallet
    def dump_keys(self):
        for key in self.keys:
            key.print_me()

        return self.keys
```

```
# EXERCISE 2.1: Creates a new bitcoin address
# Adds the new keypair (an object from rsa_key class) at the end of
# self.keys list
# fixed is an optional parameter that must be passed as the second
# argument to UOC_RSA_KeyGenerate calls
# returns the new address
def generate_new_address(self, fixed=None):
    addr = ""
    #### IMPLEMENTATION GOES HERE ####
     ##################################
    print_level(1,"generate_new_address output: " + addr)
    return addr

 # EXERCISE 2.2: Gets the wallet's balance taking into account
 # the given block_chain
 # block_chain: current block chain
 # returns a dictionary with the new balance. Dictionary keys are the
 # bitcoin addresses associated with the keys in the key storage. Value
 # for each key corresponds to the available import for that address.
 def get_account_balance(self,block_chain):
    balance = {}
    #### IMPLEMENTATION GOES HERE ####
    ##################################
    print_level(1,"get_account_balance: " + str( balance ) )
    return balance


 # EXERCISE 2.3: Creates a payment transaction using available funds
 # block_chain: current block chain
 # addr_dest: destination bitcoin address
 # returns a transaction structure with the payment or None,
 # if there are not enough funds
 def make_payment(self, block_chain, dest_addr):
    tx = transaction_struct()
    #### IMPLEMENTATION GOES HERE ####
    ##################################
    return tx
```

L'exercici consisteix en implementar els 3 mètodes que falten per implementar, és a dir, `generate_new_address()`, `get_account_balance()` i `make_payment()`.

1. Programeu una funció que generi una nova adreça de bitcoins. **(0,5 punts)**

   La funció haurà de crear un parell de claus RSA noves, que s'afegiran al final de la llista que representa el magatzem de claus, i retornarà l'adreça creada. La funció no rebrà cap argument.

   La funció retornarà la nova adreça generada.

   Nota 1: Podeu fer servir la funció `UOC_RSA_KeyGenerate` per generar les claus RSA.

   Nota 2: El mètode `generate_new_address` rep el paràmetre opcional `fixed`. Aquest paràmetre serveix per afegir un parell de claus RSA ja calculades al magatzem de claus de la classe. No cal que manipuleu el paràmetre manualment, només cal passar-lo com a segon argument a la funció `UOC_RSA_KeyGenerate`.

   Nota 3: Les claus del magatzem de claus han d'estar en format `rsa_key`.

2. Programeu una funció que retorni el balanç actual del bitlleter de bitcoins. **(2 punts).**

   La funció recorrerà la cadena de blocs i, tenint en compte les adreces associades a les claus que hi ha desades al magatzem de claus, retornarà el balanç actual del moneder de bitcoins.

   La funció rebrà com a paràmetre:

   a) `block_chain`: la cadena de blocs actual

   La funció retornarà el balanç del bitlleter desgloçat, és a dir, un diccionari que contingui l'import (en bitcoins) disponible a cada adreça. Si la cadena de blocs que rep com a paràmetre és invàlida, la funció retornarà `None`.

3. Programeu una funció que realitzi un pagament en bitcoins. **(1,5 punts)**

   La funció tindrà en compte el balanç actual del moneder de bitcoins i realitzarà el pagament, fent servir primer les adreces més antigues que continguin bitcoins.

   La funció rebrà com a arguments la cadena de blocs i l'adreça de destinació del pagament.

   a) `block_chain`: estat actual de la cadena de blocs.

   b) `dest_addr`: adreça de destí dels bitcoins.

Si el pagament és possible, la funció retornarà una estructura de dades de la transacció amb el format de transacció indicat anteriorment. En cas contrari, la funció retornarà `None`.

Exercici 3 (1 punt)

En aquest exercici implementarem la funció per incloure un nou bloc a la cadena de blocs i, per tant, generar nous bitcoins. En l'argot dels bitcoins aquesta operació està inclosa dins del procés anomenat *mining*.

1. Implementeu una funció que generi i afegeixi un nou bloc a la cadena de blocs. La funció haurà de validar que la cadena de blocs sigui correcta abans d'incloure-hi el següent bloc. **(1 punt)**

   La funció rebrà com a paràmetres una llista amb les transaccions a incloure en el bloc, l'adreça del compte on s'ha de transferir la recompensa del bloc, la cadena de blocs actual i el target del moment.

   a) `block_transactions`: llista amb totes les transaccions a incloure en el bloc.

   b) `address`: adreça bitcoin on s'ha de realitzar l'ingrés de la recompensa.

   c) `block_chain`: cadena de blocs.

   d) `target`: valor del target del bloc.

   La funció retornarà la cadena modificada, afegint-hi un nou bloc amb les transaccions indicades i enviant la recompensa de generació a l'adreça especificada.

Format i data de lliurament

La data màxima de lliurament de la pràctica és el 12/06/2014 (a les 24 hores).

Juntament amb l'enunciat de la pràctica trobareu l'esquelet de la mateixa en format SAGE notebook worksheet (extensió .sws). Aquest mateix fitxer és el que heu de lliurar un cop hi codifiqueu totes les funcions.

En aquest esquelet també hi trobareu inclosos els jocs de proves dels diferents apartats. Tal i com s'esmenta en el fitxer sws, **no es pot modificar cap part del fitxer corresponent al joc de proves**. Això vol dir que heu de respectar el nom de les funcions i variables que s'han definit en aquest esquelet.

- El lliurament de la pràctica constarà de d'**un únic fitxer** SAGE worksheet (extensió sws) on heu inclòs la vostra implementació.