Adatbiztonság és kriptográfia jegyzőkönyv
1. gyakorlat

Csutak Balázs

2018. 03. 08

Tartalomjegyzék

1. Bevezető

1.1. Kriptovaluták

A kriptovaluta olyan elektronikus fizetőeszköz ("virtuális pénz"), mely kriptográfiai eszközök alkalmazásával biztonságos fizetést tesz lehetővé egymással kapcsolatban álló felhasználók számára. A megszokott valutával szemben, mely definíció szerint egy állam törvényes fizetőeszköze, és amelynek kibocsájtását és forgalmát a megfelelő állami szerv ellenőrzi, a kriptovaluta nem kapcsolódik be a hagyományos bankrendszerbe, hanem decentralizált módon, központi felügyelet nélkül, a hagyományos pénztőlforgalomtól szinte függetlenül működik.

Jelen jegyzőkönyvben röviden ismertetem a kriptovaluták, ezen belül a nagy népszerűségnek örvendő Bitcoin működését, a rendszer felépítésének technikai megoldásait és kriptográfiai hátterét. Bár a kriptovaluták gazdasági és jogi vonatkozásai, úgy mint a kibocsájtott pénzmennyiség mögötti fedezet, értékállóság, avagy a lebegtetett árfolyamrendszerrel való kapcsolat is rendkívül érdekes kérdések, ezekre jelen jegyzőkönyvben nem térek ki.

A kriptovaluta lényege, hogy központi szabályozó- illetve rendszerfelügyelő szerv nélkül, a felhasználók által alkotott peer-to-peer hálózat csomópontjain nyilvántartott elosztott adatbázisban tárolt tranzakciós lista alapján működik. A felhasznált kriptográfiai eszközök biztosítják a tranzakciók hitelességét és eredetiségét, illetve rendkívüli módon megnehezítik (kvázi lehetetlenné teszik) az adatbázis haszonszerzési célzattal történő manipulálását egy felhasználó vagy felhasználók egy csoportja által.

Digitális aláírás

A kriptovaluták rendszerében egy felhasználó a digitális aláírásával képes igazolni magát és fizetési szándékát. Ha tranzakciót szeretne kezdeményezni, ezt a kérést digitálisan aláírva továbbítja a hálózat csomópontjai felé. A tranzakció attól a pillanattól kezdve érvényes, ahogy bekerült az elosztott adatbázisba. Ez egyúttal anonimitást is biztosít: bár a tranzakció léte nyilvános, csak a résztvevők digitális aláírása látható. Ha az aláírást használó fizikai személy nem fedi fel kilétét, a pénz forrása és célja nem azonosítható.

Hash függvények

A jelenlegi kriptovaluták nagy része, a töretlen népszerűségnek örvendő Bitcoin is proof-of-work rendszeren alapul. A koncepció lényege, hogy az adatbázisba bekerülő adatok hitelességét az adat beírásához szükséges nagy mennyiségű munka igazolja. Ennek a munkának a szükségességét biztosítják a hash függvények.

Blockchain

A blockhain az elosztott adatbázis tárolási módja, lényege, hogy az adatok - kriptovaluták esetében a tranzakciók - egymásra épülő blokkokban kerülnek tárolásra. A megközelítés

biztosítja, hogy az adatbázis kapcsolódó elemei kronológiai sorrendben követik egymást, múltbeli tranzakciók nem fordíthatóak vissza, és megakadályozza az olyan csalárdlehetőségeket, mint ugyanazon pénz két alkalommal történő elköltése.

Az alábbiakban részletesen bemutatom a Bitcoin rendszerben alkalmazott digitális aláírást, hash függvényt és blockhain építés szabályait.

1.2. Hash függvények, SHA256

A hash olyan függvény, mely tetszőleges bináris bemenethez egy fix hosszúságú, determinisztikusan véletlenszerű kimenetet állít elő. A függvény egyirányú, így a kimenetből lehetetlen a bemenetre következtetni - egy adott kimenethez passzoló bemenetet megtalálni csak kimerítő kereséssel lehet. Hasonlóan nehéz, azaz kvázi lehetetlen találni két ugyanolyan hash-el rendelkező bemenetet.

Az SHA-256 az amerikai nemzetbiztonsági ügynökség (NSA) által kifejlesztett és a FIPS által szabványosított, kriptográfiai célokra használható hash függvény. Ahogy neve is sugallja, tetszőleges bemenethez 256 bit hosszúságú lenyomatot képez. Egyetlen bit megváltozása a bemeneten teljesen más hash-hez vezet, így ha adott kimenethez próbálunk ehhez illeszkedő bemenetet találni, a tesztbemenet hashéből nem látjuk, hogy mennyire járunk közel a célhoz.

Ez utóbbi tulajdonságnak köszönhetően képes a hash függvény biztosítani, hogy csak a megfelelő mennyiségű munkával lehessen új adatot írni az adatbázisba: minden adatblokk beillesztésénél, a bemenet egy kis részének variálásával kell elérni, hogy a blokk lenyomata előre meghatározott számú 0-bittel kezdődjön. Mivel ez az eredmény csak kimerítő kereséssel, heurisztika nélkül végezhető, a munka számítási komplexitása $\mathcal{O}(2^n)$, ahol n a szükséges 0-bitek száma. Jelenleg a Bitcoin rendszer 64 nullbitet ír elő az új blokkok számára, mely mennyiség folyamatosan növekszik. Utánaszámolva, ez a komplexitás komoly erőforrásokkal rendelkező mainframe gépek számára is több perc, vagy akár több óra számolást jelent.

1.3. Digitális aláírás, RSA

A digitális aláírás célja, hogy egy adott adatcsomag eredete kétség nélkül ellenőrizhető legyen. Napjainkban a legelterjedtebb digitális aláírás-rendszerek asszimetrikus kulcsú titkosításon, nevezetesen az RSA algoritmuson alapulnak.

Az RSA lényege, hogy minden felhasználó rendelkezik egy ún. RSA kulcspárral: egy publikus és egy privát kriptográfiai kulccsal. A kulcsok közt matematikai kapcsolat van: ugyanazon két prímszám segítségével lettek előállítva. Ez a kapcsolat biztosítja, hogy az egyik kulccsal végzett titkosítási művelet visszafejtése csak a másik kulccsal lehetséges. Az algoritmus erejét az adja, hogy az üzenet kulcs nélküli visszafejtéséhez, avagy a privát kulcs publikus kulcsból való előállításához a támadó képes kell legyen "nagy számok" (napjainkban ez $2^{1024} - 2^{2048}$ nagyságrendet jelent) prímtényezős felbontására, ami megfelelően generált prímek esetén NP-teljes probléma.

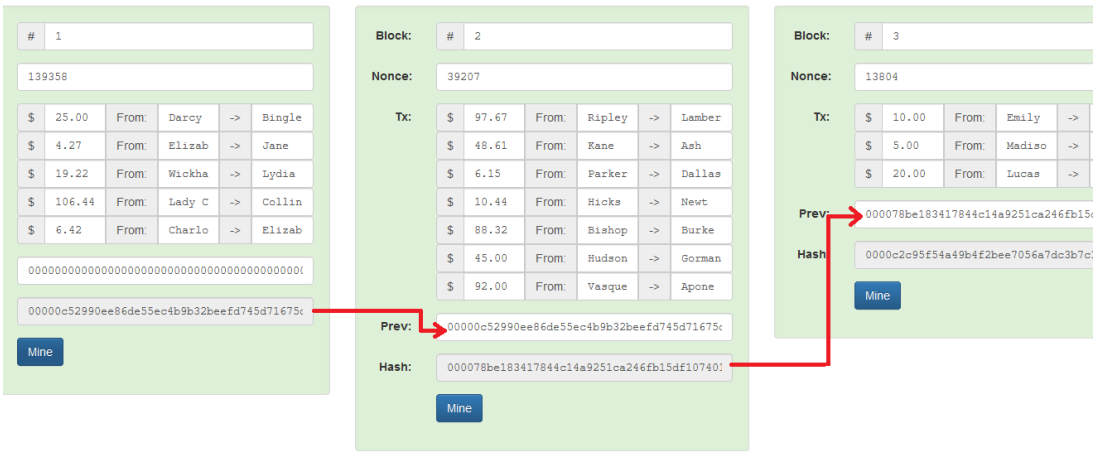
A módszer a következőképpen használható digitális aláírásra:

1. Az üzenet feladója kiszámolja az aláírni kívánt adat hash értékét
2. A kapott hash értéket titkosítja a privát RSA kulcsával, és az üzenet mellé csatolja
3. A címzett ugyancsak kiszámolja az adat hash értékét
4. A címzett a feladó publikus kulcsával visszafejti az üzenethez csatolt aláírást, és összehasonlítja a kiszámolt hash-el
5. Ha a két érték megegyezik, az üzenetet biztosan a publikus kulcs privát párjának birtokosa küldte, hiszen akár az üzenet módosítása, akár a nem megfelelő privát kulccsal való titkosítás esetén más értékeket kapunk.

A módszer természetesen csak akkor működik, ha ismerjük az ellenőrizni kívánt feladó publikus kulcsát. Mivel a kriptovaluta rendszerben mindenkit az aláírása azonosít (pontosabban az aláíráshoz használt RSA kilcspár publikus része), ez egyáltalán nem jelent problémát.

1.4. Blockchain működése

A kriptovalutáknál alkalmazott blockchain lényege, hogy a blokkok belsejébe a tranzakciók kerülnek. Mindegyik blokk tartalmazza az előző blokk hash értékét, így biztosítva a rákövetkezőséget. A blokk beillesztéséhez szükséges meghatározni egy ún. nonce értéket, amivel a block hash értéke befolyásolható. A megfelelő mennyiségű 0-ával kezdődő hash-el rendelkező blokk bekerülhet a láncba, a hálózat elemei elfogadják érvényes adatként.



1. ábra. Blocks of a cryptocurrency blockchain, with links illustrated between them [7]

Mivel egy jó ábra gyakran többet mond minden szónál, itt látható az egymásra épülő, tranzakciókat tartalmazó blokkok kapcsolata a blockchainen belül. A hash függvény tulajdonsága biztosítja, hogy a lánc blokkjai a ráépülések elrontása nélkül nem kicserélhetők, azoknak egyetlen bitje sem módosítható.

2. Elvégzett feladatok

2.1. Előkészületek - python és pycryptodome

A tantárgy wiki oldalán szereplő előkészületeket a gyakorlat előtt elvégeztem. Ezek az alábbi feladatok voltak:

- Python 3.6.4 telepítése és PATH-hoz való hozzáadása
- PyCryptodome könyvtár telepítése
- Ismerkedés a Python nyelvvel
- A gyakorlathoz kiadott kód letöltése, előkészítése a munkához

A gyakorlat további részében a megadott kódot egészítettük ki, a gyakorlatvezető útmutatásai és a hivatkozott netes források alapján.

2.2. Kulcspár generálása és mentése

```
1     @staticmethod
2     def generateΓ`_key():
3
4         key = RSA.generate(2048)
5         encryptedΓ`_key = key.exportKey('PEM');
6
7         fileΓ`_out = open(Γ`¨rsaΓ`_key.PEMΓ`¨, Γ`¨wbΓ`¨)
8         fileΓ`_out.write(encryptedΓ`_key)
```

A kódsorok a következőket végzik:

- statikus metódusként deklarálom a kulcsgenerálást a Blockchain nevű python osztályba. A metódust tipikusan csak egyszer, a folyamat legelején fogjuk meghívni.
- generálok egy RSA kulcspárt, és a `key` python-objektumba mentem
- a privát kulcsot PEM formátumban [3] elmentem az `encrypted_key` változóba. A változó elnevezésének háttere, hogy a kulcsot tipikusan egy felhasználói jelszóból generált szimmetrikus kulccsal titkosítva szokás tárolni, így nehezítve a potenciális kulcslopást. Ebben a példafeladatban ezt a lépést kihagytam.
- megnyitom az `'rsa_key.PEM'` fájlt írásra, bináris formában. Mivel a PEM base-64 kódolású, ez a megszorítás nem annyira indokolt - text based kiírással is működik, de így kérte a feladat...
- a kulcsot kiírom a fájlba

2.3. A kulcs visszaolvasása

Ezt a kódrészletet a [6] alapján implementáltam, lépései lényegében az előző rész lépései fordított sorrendben: a fájlt megnyitom olvasásra, a kulcsot pedig PEM formátumból a megfelelő python-objektumba importálom. Mivel a kimentésnél nem adtam meg szimmetrikus titkosítási kulcsgeneráláshoz használható felhasználói jelszót, a bemenetet nem kell visszafejtenem, a kulcs plain-text formában van a fájlban.

```
1     @staticmethod
2     def readΓ`_key():
3         encodedΓ`_key = open(Γ`¨rsaΓ`_key.PEMΓ`¨, Γ`¨rbΓ`¨).read()
4         key = RSA.importΓ`_key(encodedΓ`_key)
5         return key
```

2.4. Hashelés pythonban

Ezt a függvényt már megírva kaptam, a feladat csak a működés megértése és dokumentálása volt:

```
1     @staticmethod
2     def hash(data):
3         hash = SHA256.new()
4         encodedΓ`_data = json.dumps(data, sortΓ`_keys=True).encode()
5         hash.update(encodedΓ`_data)
6         return hash
```

A kódsorok a következőket végzik:

- létrehozok egy SHA256 hasheléshez használható objektumot (PyCryptodome csomag része)
- mivel nem a python objektumot, csupán annak tartalmát szeretném hashelni, ezt a python nyelv dokumentációja alapján exportálom JSON formátumba [4]
- kiszámolom a JSON formátumba kimentett tartalom SHA256 hash értékét

2.5. Digitális aláírás

A fent részletezett módon, a digitális aláírás során lényegében titkosítjuk az aláírni kívánt adat hash értékét a privát RSA kulccsal. A titkosításhoz a feladatkiírásnak megfelelően a PKCS1_v1_5 szabványcsaládban leírtak szerint végezzük [5].

```
1     def signΓ`_data(self, data):
2         hash = self.hash(data)
3         signature=PKCS1Γ`_v1Γ`_5.new(self.privateΓ`_key).sign(hash)
4         return b64encode(signature)
```

A kódsorok a következőket végzik:

- az előbb dokumentált függvény segítségével kiszámolom az aláírni kívánt adatot tartalmazó objektum értékének hash értékét
- ezt aláírom a Blockhain objektum konstruktorában a már ismertetett `read_key` metódussal beolvasott privát kulccsal, a szabvány szerinti kriptográfiai műveletekre alkalmas PyCryptodome objektum (`PKCS1_v1_5`) segítségével
- az aláírást a későbbi kezelhetőség (fájlba írás, stb.) érdekében base64-kódolással elmentem

2.6. Aláírás ellenőrzése

Az aláírást a 1.3 részben leírtak alapján ellenőrzöm. A feladatkiírásnak megfelelően, az ellenőrzést is a PKCS1_v1_5 szabványcsaládban leírtak szerint végezzük [5].

```
1     def verify(self, data, signature, pubΓ`_key):
2         signature = b64decode(signature)
3         hash = Blockchain.hash(data)
4         verifier = PKCS1Γ`_v1Γ`_5.new(pubΓ`_key)
5         if verifier.verify(hash, signature):
6             return True
7         else:
8             return False
```

A kódsorok a következőket végzik:

- A base64 formában kapott aláírást visszaalakítom binárisba, hiszen a kriptográfiai fügvények így tudják kezelni.
- Kiszámolom az ellenőrizni kívánt adat hash értékét a már ismertetett hash függvénnyel.
- Az aláírást egy szabvány szerinti kriptográfiai műveletekre alkalmas objektummal (`PKCS1_v1_5`) visszafejtem.
- Ellenőrzöm, hogy az imént számolt hash értéket kaptam-e. Ha igen, akkor az adatot valóban a publikus kulcs párjának birtokosa írta alá.

2.7. A kód tesztelése

A megírt függvény segítségével generálok és fájlba mentek egy RSA kulcspárt:

```
Blockchain.generateΓ`_key()
```

Ezután a következő műveleteket végzem:

```
bc = Blockchain()
data = Γ`¨Hello World!Γ`¨
signature = bc.signΓ`_data(data)
data = Γ`¨Hello World!Γ`¨
print(bc.verify(data, signature, bc.publicΓ`_key))
data = Γ`¨Hello Laci!Γ`¨
print(bc.verify(data, signature, bc.publicΓ`_key))
```

Az implemntált algoritmusok teszteléséhez a fenti kóddal létrehozom a `Blockchain` osztály egy példányát, mely konstruktorban beolvassa a gyakorlat elején generált kucspárt, felhasználva a már bemutatott `read_key` metódust.

```
def Γ`_Γ`_initΓ`_Γ`_(self):
    key = self.readΓ`_key()
    self.privateΓ`_key = key
    self.publicΓ`_key= key.publickey()
```

A műveletek kimenete:

```
True
False
```

Az aláírt üzenet (”Hello World!”) megegyezik az első tesztüzenettel, viszont különbözik a másodiktól (”Hello Laci!”).

3. Következtetések

A jegyzőkönyvben összefoglaltam a kriptovaluták, azon belül a Bitcoin működését, felhasználva az előadáson készített saját jegyzeteimet, a tantárgy wiki oldalán levő diasorokat, valamint több, a hivatkozások részben található internetes forrást. Részlesen leírtam és illusztráltam, hogyan működik a blockchain, milyen szerepük van a hash függvényeknek a lánc építésében, és hogyan igazolja a digitális aláírás egy tranzakció eredetiségét.

A gyakorlat idejében az összes kiírt feladatot sikerült teljesíteni. A jegyzőkönyvbe illesztett kódrészletekkel bemutattam a kért kriptográfiai műveletek Python/PyCryptodome

implementációját, és egy rövid tesztkóddal ellenőriztem ezek helyességét. A teljes kód a [8] címen megtekinthető és letölthető futtatható formában.

Összegezve, a kriptovaluták és a Bitcoin működésének alapjait sikerült megérteni, dokumentálni, és néhány alapvető funkciót bemutató jelleggel lekódolni.

Hivatkozások

[1] A Bitcoin hivatalos weboldala, azon elhelyezett tutorial, tudásbázis, hírek és haladó felhasználóknak szóló anyagok:
`https://www.bitcoin.com/getting-started`
`https://www.bitcoin.com/bitcoinBasicsCourse`
`https://www.bitcoin.com/guides`
`https://www.bitcoin.com/news`
`https://www.bitcoin.com/bitcoin-mining`

[2] A tantárgy Wiki oldala: Adatbázis és Kriptográfia

[3] Privacy-enhanced Electronic Mail, kriptográfiai kulcsok biztonságos és nyomtatható formban történő tárolására kidolgozott fájlformátum: `https://en.wikipedia.org/wiki/Privacy-enhanced_Electronic_Mail`

[4] Usage of JSON python library, exporting object contents in hierarchical way: `https://docs.python.org/2/library/json.html`

[5] Public-Key Cryptography Standards (PKCS), published by RSA laboratories in 1993: `https://tools.ietf.org/html/rfc2313`

[6] PyCryptodome használata, hivatalos példák:
`http://pycryptodome.readthedocs.io/en/latest/src/examples.html`

[7] `https://anders.com/blockchain/tokens.html`

[8] Csutak Balázs weboldala, adatbázis és kriptográfia:
`http://users.itk.ppke.hu/~csuba/Kripto`