Adatbiztonság és Kriptográfia

02. gyakorlat

jegyzőkönyv

Ekart Csaba [ZWPMKP]

2020. április 5.

Tartalomjegyzék

Előkészületek és bevezetés

A gyakorlaton a Tanár úr vezetésével átnéztük az általa írt kódot, mely az előző gyakorlaton már megírt *ITKoin01.py* folytatása volt. Az első gyakorlaton főleg az alapvető kriptográfiai függvényekkel foglalkoztunk, így létrehoztunk publikus és privát RSA kulcsokat, amelyeket fájlba mentettünk és beolvastunk, lenyomatoló függvényt készítettünk és aláírásokat hoztunk létre, illetve ezeket verifikáltuk.
Ennek folytatásaként a második gyakorlaton megismerkedtünk a blokkláncról tanultak lehetséges kód szintű megvalósításával, a valuta bányászatával, illetve a tranzakciók létrehozásának folyamatával. Ez a jegyzőkönyv ezen kódrészek működését mutatja be.

1. Az adatstrúktúra felépítése

A múltheti gyakorlaton megírt osztály fontos adattagokkal bővült ki. Ezek szerepének és felépítésének rövid bemutatása következik, röviden megjegyezve, hogy pontosan mely metódusok lesznek rájuk közvetlen hatással, illetve melyek használják fel ezeket az adatokat. Ezen metódusok részletes leírása a következő fejezetben található.

self.chain

Az osztály `self.chain` adattagjában tároljuk a magát a blokkláncot. Ez egy listát jelent, amely a létrehozott blokkokat fogja tartalmazni. Ezt később a `self.save_list()` metódus segítségével fájlba fogjuk írni, majd a `self.load_chain()`-nel beolvashatunk. A beolvasás során érdemes ellenőrzéseket alkalmazni, melyben megvizsgáljuk, hogy a blokklánc érvényes e. Lánc szinten a blokkok egymásra épülését kell majd megvizsgálni, de blokk szinten is célszerű ellenőrzéseket létrehozni. Ezek pontos megtervezése, megvalósítása, illetve dokumentálása a következő jegyzőkönyv témája lesz.

A lánc változót a betöltést követően többször is használjuk, illetve módosítjuk

- `self.find_unspent_outputs()`: ebben a metódusban közvetlenül a blokklánc alapján generáljuk az el nem költött outputok listáját, melyben a következőkben részletesebben is szó lesz.
- `self.mine()`: minden egyes kibányászott blokkot a lánc végéhez fűzünk. A metódus működésének részletes leírása a következő fejezetben található.

self.pending_transactions

A `self.pending_transactions` listában azon tranzakciókat tároljuk, melyek még nincsenek blokkban, hanem feldolgozásra várnak. Bitcoinhoz képest különbség lesz, hogy ezek között nem lehet majd válogatni, hanem mindegyiket bele fogjuk tenni a soron következő blokkba. Hasonlóan a blokkláncnhoz ezt is kimentjük fájlba, majd onnan visszaolvassuk. Szintén következő jegyzőkönyv témája ezeknek az ellenőrzése beolvasáskor, minthogy a felhasználói jogosultság ellenőrzése, hogy bizonyos outputot inputként használt egy tranzakcióban, az összegek rendben vannak e stb.

A legfontosabb eljárások, melyekben ezt a listát használjuk

- `self.mine()`: ebben a metódusban épül fel az új blokk, mely pont a lista által tartalmazott fel nem dolgozott tranzakciókat fogja elbírálni, majd beépíteni a soron következő blokkba.
- `self.generate_first_block()`: ebben a metódusban hozzuk létre az első blokkját a blokkláncnak. Itt a kezdeti tranzakciók kerülnek be a listába, melyek `self.initial_csaposhi_offering` változóban meghatározott értéket fogják eljuttatni a valuta felhasználóinak. Részletes leírása a következő fejezetben kifejtésre kerül.
- `selg.new_transaction()`: természetesen a kezdeti tranzakciókhoz hasonlóan a későbbiek is létrehozásuk után ebbe a listába töltődnek. A leírása szintén a második fejezetben található.

self.unspent_outputs

Az eddigi adatokból hozzuk létre. Olyan lista, mely azokat az output összegeket tartalmazza, amelyek még elkölthetőek. Praktikusan azon outputok, melyeket nem használtunk még fel más tranzakciók inputjában. Erről a `self.find_unspent_output()` függvény gondoskodik. Ebben a metódusban közbetlenül a blokklánc összes blokkján végighaladva követjük le, hogy mely outputok felhasználhatóak. Fontos, hogy az első blokkot külön kezeljük, mert az nem használ fel korábbi outputot, itt a "semmiből" jön létre a pénz. A szerkezet, amit létrehozunk egy adathármas lesz, mely a tranzakció azonosítójából, egy sorszámból, valamint az outputból áll. Az output tipikusan két összeg, a valutamennyiség amit fizettem, valamint a magamnak utalt visszajáró. Működését tekintve, először az összes tranzakciót felvesszük a listába, majd ezt követően egy ciklusban eltávolítjuk azokat, amik outputjai már el lettek költve valaki más inputjában.

self.my_unspent_outputs

Ez az adatmező rendkívül hasonlóan épül fel a `self.unspent_outputs`-hoz, azzal a különbséggel, hogy csak azokat az elemeket tartalmazza, ahol az output hozzám került, tehát általam elkölthető. Ezt a listát ugyanott töltjük fel, ahol a `self.unspent_outputs`-t is tettük, csak egy extra lépésben megvizsgáljuk, hogy az outputhoz tartozó recipient ID-ja megegyezik e a mienkkel. Később - szintén hasonlóan az előzőhöz - eltávolítjuk azokat az elemeket, amelyek fel lettek már használva más tranzakció inputjában.

A benne tárolt információt használjuk például a `self.new_transaction()` metódusban, ahol addig veszünk ki a listából elemeket, amíg azok összege alkalmassá nem válik egy előre meg határozott összeg kifizetésére.

self.ITKoin_users és self.my_id

A rendszerben minden résztvevőt egy azonosító alapján tartunk nyilván. Ezeket az azonosítókat a publikus kulcsukból hozzuk létre lenyomatolással és hexakódolással, majd fálba írjuk. A felhasználó létrehozása úgy történik, hogy generálunk egy kulcspárt, és elmentjük a publikus és magán kulcsot 1-1 fájlba, majd a publikus kulcsból a fent említett módon megalkotjuk az azonosítót is. A kódban két felhasználót használtunk, az egyikkel magunkra hivatkoztunk a másik pedig egy partner / tesztalany volt. Az adatmezőben így nem az azonosítók listája található, hanem azon fájlok nevei, melyek ezeket az azonosítókat tartalmazzák.

A `self.ITKoin_users`-ben tárolt felhasználók adatait akkor használjuk fel, mikor a `self.generate_first_block()` során mindenki számára létrehozunk egy tranzakciót, melyben megkapják az előre meghatározott kezdeti tőkét.

A `self.my_id` jelöli a saját azonosítónkat, melyet a privát kulcs betöltésekor tárolunk el. Ezt később a saját fel nem használt outputjaink meghatározásánál, illetve új tranzakció létrehozásakor használjuk fel.

2. A legfontosabb eljárások működése

generate_first_block()

Ez a metódus felel a blokklánc első blokkjának legyártásáért. A cél, hogy minden felhasználó megkapja az `self.initial_csaposhi_offering`-et. Ezt az eljárást egyszer hívjuk meg, miután létrehozzuk a valutánkat. Először végighaladunk az összes felhasználón (akiket a korábban említett `self.ITKoin_users`-ben tárolunk), és beolvassuk azonosítókat. Ezt követően elvégzünk egy extra ellenőrzést, melyben leellenőrizzük, hogy véletlenül sem szerepel kétszer ugyanaz a recipient. Erre azért van szükség, mert ha így lenne, akkor mivel az összeg is megegyezik (ez a kezdeti összeg), akkor a később létrejövő tranzakciós azonosító is ugyanaz lenne, mert ezen paraméterekből generáljuk azt. Ennek a gyakorlatban az esélye meglehetősen alacsony, de nem árt biztosra menni.

Ha mindent rendben találunk létrehozzuk a szükséges kezdeti tranzakciókat, melyek outputjaként minden felhasználó rendelkezni fog a kezdő összeggel. A tranzakciókat ellátjuk egy azonosítóval (amely önmaga lenyomata). Ezt követően kibányászhatjuk az első blokkot a `self.mine()` eljárás segítségével.

self.mine()

A `self.mine()` függvényt a bányászok használják, hogy felépítsék a blokkokat és felfűzzék a láncra. Az eljárásban külön választjuk az első blokk és a többi bányászatának folyamatát. Erre azért van szükség, mert az első blokk előtt nincs még más, amivel össze kéne fűzni, így az előző block headerjének azonosítóját - a `previous_block_header_hash`-t egy tetszőleges hexa értékre állítjuk, pl. "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff". Minden más esetben megvizsgáljuk a lánc előző, azaz utolsó elemét, és ennek a hashét fogjuk felhasználni.

Ezután történik magának a bányászásnak a folyamata. Ennek során létrehozzuk a soron következő blokk headerjét. Ezt egy növekvő változóból, az előző header lenyomatából, illetve a tranzakciók lenyomatából hozzuk létre. Addig kell dolgoznunk, míg az ebből létrehozott header meg nem felel valami kritériumnak, mely a mi esetünkben az, hogy 4 db. 0-val kezdődjön. Minden sikertelen próbálkozás után növeljük a változónkat. A gyakorlaton láthattuk, hogy a próbálkozások száma, akár 100.000 fölé is felszökkenhet, ilyenkor futtatásnál a lassulás is érezhető volt.

Ha sikerült sikeresen létrehoznunk a lenyomatot, hogy az megfeleljen a szükséges kritériumnak, segítségével felépítjük a soron következő blokkot. Ez az imént készített headerből, illetve a tranzakciókból készített szótár objektum lesz. A létrehozott blokk immár alkalmas, hogy a blokkláncot tartalmazó `self.chain` tagot kibővíthessük vele.

A blokk létrehozása után kiürítésre kerül a `self.pending_transactions`, hiszen ezen tranzakciókat az imént építettük be a blokkba.

self.new_transaction()

Az utolsó függvény, melyet a gyakorlaton részletesen áttekintettünk a `self.new_transaction()` volt. Ezt a metódust a felhasználók használják, amennyiben tranzakciót hajtanak végre. Lényegében összeszedünk annyi olyan felhasználható outputot, amit inputként felhasználva kifizethetjük az outputot. A függvény két paramétert vár: mekkora összeget kívánunk küldeni, és ki a címzett. Ha sikerült összeszednünk annyi outputot, amennyi szükséges az összeg kifizetéséhez, akkor ezeket az elemeket, amik jelenleg a tranzakció azonosító, sorszám, összeg hármast tartalmazzák kiegészítjük két további adattaggal: az azonosítóból létrehozott aláírással, illetve a publikus kulcsunkkal. Amennyiben az összeg nem pontos (és ez nagy valószínűséggel így lesz) egy további outputtal bővítjük a tranzakciót: a magunknak visszautalandó visszajáróval. Ezt követően a tranzakciót létrehozzuk az input és output értékekből, valamint a lenyomatából képzett azonosítójával (hasonlóan, ahogy az első blokk létrehozásánál tettük).

Ebben a metódusban ellenőrzéseket nem végzünk, a tranzakció helyességének ellenőrzését úgyis a bányászok fogják elvégezni, ez biztosítja, hogy a láncba kerülő blokkok csak helyes tranzakciókat tartalmaznak.