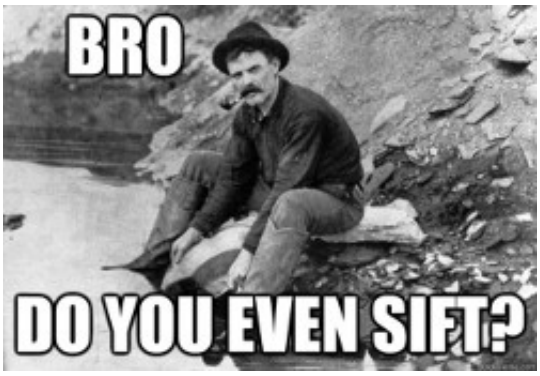
Vorgurakendused 2 prax 2 2019 kevad

Allikas: Lambda



Sisukord

Ülesanne

Teise praktikumi jaoks võid valida ühe kolmest ülesandest:

- Minimaalse funktsioneeriva distributed ledgeri ehitamine esimese praksi edasiarendusena. See on analoogiline eelmise aasta ülesandega.

- Teine variant: videostriimingu näiterakendus. See rakendus peab suutma võtta vastu brauserist või mobiilist käivitatud videoülesvõtet ja (a) salvestama seda serveris olevasse faili, mida saab pärast läbi brauseri tagantjärgi vaadata (b) võimaldama läbi teises masinas oleva brauseri seda videoülesvõtet jooksvalt vaadata.

- Kolmas variant on sinu enda valitud teema, mis on õppejõuga eelnevalt kooskõlastatud: et kas sisuliselt sobib ja kas on mõistliku keerukusega. See on mõttekas valik, kui sul on juba mõte mõne rakendusliku süsteemi ise-ehitamiseks ja sisu klapib meie kursusega.

Esimene variant: terve distributed ledger

See praktikum tuleb ehitada esimese ledgeri-praktikumi edasiarendusena. Eesmärk on saada käima lihtsustatud versioon bitcoini-tüüpi distributed ledgerist. Keerukaid nüansse - mis päris-bitcoini puhul vajalikud - realiseerida ei ole vaja.

Mis meil on - kokkuvõtlikult - vaja esimesele praksile lisada:

- Transaktsiooni sisestamine ja konkreetses formaadis edastamine: kes kannab kellele kui palju raha üle
- Digiallkirjastamine, avalikud ja salajased võtmed
- Transaktsioonide kokkukogumine blokiks ja bloki ülekontrollimine
- Bloki merkle juure ehitamine (salvestame lõpuks ainult tipmise hashi puust)
- Bloki kaevandamine: leia bloki hash, mis algaks N nulliga, katsetades eri nonce. Sea N nii, et kaevandamine võtaks ca 5 sek.
- Mitme paralleelbloki olemasolul pikema ahela või parema bloki valimine
- Loodud bloki automaatselt teistele edasisaatmine

Eeldame, et sinu programm juba kuulab ja salvestab ja saadab edasi talle saadetud transaktsioone ja blokke.

Konkreetseid detaile ei kirjuta me siin praksis kohustuslikuna ette, kuid järgmisena on toodud soovitusi ja nõuandeid.

Päris hea arusaamist tekitava veebidemo leiad siit (https://anders.com/blockchain/public-private-keys/blockchain.html): kindlasti vaata ka ülalt menüüst keys/signatures/transaction demosid.

Transaktsioon

Võiks olla ilma allkirjata kujul

```
{"from": "sdasd212121", "to": "dsda9sdasdas9", "sum": 0.01, "timestamp": "2012-04-21T18:25:43-05:00"}
```

kus timestamp on iso kujul. Seejuures transaktsioon tuleb enne signeerimist stringiks teisenda mingil ühesel universaalsel kujul (a la keyd konkreetses järjekorraks ja tühikud key-value paaride vahelt eemaldatud) ja allkirjaga kujul

```
{"signature":"alasasasp11", "transaction": {"from": "sdasd212121", "to": "dsda9sdasdas9", "sum": 0.01, "timestamp": "2012-04-21T18:25:43-05:00"}}
```

Mis on see "from" ja "to" väärtus, ehk, kuidas kasutajaid/kontosid identifitseerida? Las need olla lihtsalt kasutajate avalikud võtmed, millest me järgmisena räägime.

Sinu programm peaks olema võimeline tegema transaktsioone: näiteks nii, et käivitad ta käsurealt ja annad ette transaktsiooni jsoni, tema allkirjastab ja saadab allkirjaga edasi.

Või siis ta loeb iga N sekundi tagant faili, kuhu saad kirjutada transaktsioonide jsone.

Või mõnel muul sulle mugaval moel.

Allkirjastatud transaktsioon tuleks teistele progedele edasi saata.

Digiallkirjastamine, avalikud ja salajased võtmed

Sul on oma "konto" tegemiseks vaja genereerida mingi public-key krüptosüsteemi jaoks avaliku ja salajase võtme paar. Muidugi on oluline, et kõik võrgustiku osalised kasutaks sama public-key krüptosüsteemi.

Kui oled genereerinud uue avaliku võtme A ja vastava privaatvõtme P, siis, teades A-d, ei suuda keegi teine reaalselt leida sinu P-d.

Olgu sinu A avalik võti ja P sinu vastav privaatvõti. Siis krüpteeri(T,P) annab stringi K, nii et igaüks saab teha dekrüpteeri(K,A) mis annab tulemuse X ja näha, et tulemus X võrdub algse stringiga T. Samas ei suuda keegi genereerida ise stringi K (teadmata P-d), et saaksime dekrüptimisel sama tulemuse T.

Peamiselt kasutatakse kas RSA tüüpi või elliptilise kõvera tüüpi public-key krüptosüsteeme. Näiteks vana eesti id-kaart kasutas RSA-d, peale sügisest turvaauku hakati kasutama elliptilist kõverat. Bitcoin kasutab samuti elliptilist kõverat: ECDSA varianti standardse elliptise kurviga “secp256k1”. Meie praksis kasuta lihtsalt sellist süsteemi, mille saad kergemini käima ja mida on sul kergem kasutada.

Kust leida public key krüptograafia teeke?

- Pythoni jaoks on hea variant näiteks python-ecdsa (https://github.com/warner/python-ecdsa)
- Kõige mainstreamim universaalne teek (koos käsurea-utiliitidega) on openssl (https://www.openssl.org/)
- Pythonis saad openssl-i kasutada näiteks pyopenssl (https://pyopenssl.org/en/stable/) wrapperiga või käsurea-utiliitidega.

Hea mõte on wikipediast lisaks lugeda public key cryptography (https://en.wikipedia.org/wiki/Public-key_cryptography) ja digital signature (https://en.wikipedia.org/wiki/Digital_signature).

NB! Põhimõtteliselt võid digiallkirja (ehk, privaatvõtmega krüpteeritud stringi) arvutada kas otse transaktsiooni stringi-kujust või siis selle stringi-kuju SHA hashist. Miks SHA hashe kasutatakse? Sest pika stringi krüpteerimine on aeglane ja SHA hash on suhteliselt lühike ka pikkade algstringide korral.

Transaktsioonide kokkukogumine blokiks ja blokiandmete ülekontrollimine

Ütleme, et sinuni on jõudnud N transaktsiooni. Sina tahad nad blokiks kokku panna. Kõigepealt vaata, millised transaktsioonid juba on mõnes sinule teada olevas blokis: neid ära võta.

Järgmisena peaksid kontrollima, kas kõik transaktsioonid on õieti allkirjastatud: selleks pead transaktsiooni allkirjad transaktsiooni avaliku võtmega dekrüptima ja kontrollima, kas tuleb transaktsioonistring: vaata eelmist peatükki.

Järgmisena peaksid kontrollima, kas transaktsiooni tegijal on piisavalt raha: selleks summeeri kõik talle saadetud ja tema poolt saadetud summad üle kõigi sulle teada olevate blokkide.

Järgmisena peaksid kontrollima, kas mõni transaktsioon on juba uues kokkupandavas blokis või mõnes vanas blokis olemas: kui jah, siis seda ei tohi kasutada.

Kui mõni transaktsioon on valesti allkirjastatud või raha ei jätku või on topelt, viska see transaktsioon lihtsalt ära.

Reaalse bitcoini kontrolli-algoritmi leiad siit (https://en.bitcoin.it/wiki/Protocol_rules#.22block.22_messages) ja andmestrukuurid siit (https://en.bitcoin.it/wiki/Protocol_documentation): seal on praktikumi jaoks liiga palju detaile, nii et soovitaks mitte hakata kogu seda "ametlikku" süsteemi implementeerima.

Lisa transaktsioonidele veel üks: endale nullist raha kandmine. Pane see näiteks viimaseks transaktsiooniks ja kanna endale 1 coin. Saatja identifikaatoriks pane näiteks 0. Seda transaktsiooni pole mõtet allkirjastada, jäägu see "eritransaktsiooni" jaoks tühjaks (kontrollimise käigus peaks siis vastavalt vaatama, et blokis võib olla üks selline 0-saatjaga ilma allkirjata transaktsioon ja see on ok).

Nüüd pane kontrolli edukalt läbinud transaktsioonid blokiks kokku. Selles tee näiteks key/value struktuur ehk json näiteks nii:

```
{
 "nr": bloki_number_ehk_eelmise_bloki_nr+1,
 "previous_hash": ahelas_eelmise_bloki_hash,
 "timestamp":  "2012-04-21T18:25:43-05:00",
 "nonce": "asasas11",
 "hash": "000000sasas0a0s0111",
 "creator": minu_avalik_võti,
 "merkle_root": "asasasas1w111",
 "count": number_of_transactions,
 "transactions": [{...}, ..., {....}]
}
```

Bloki Merkle juure (merkle root) arvutamine

Mõte selles, et seome kõik bloks olevad transaktsioonid nende hashide kaudu puukujuliselt üheks merkle hashiks ja ei oleks vaja hoida iga transaktsiooni hashi eraldi. Seda peab tegema kindlas järjekorras, et teised kontrollida saaks. Kõigepealt pead arvutama iga transaktsiooni hashi (selleks konverdi transaktsioon stringiks) ja siis hakka neid hashe hierarhiliselt paariviisi kokku konkateneerima ja selliselt paariviisi kokku liidetud hashidest uusi hashe arvutama.

Vaata Merkle puu kohta

- wikipedia (https://en.wikipedia.org/wiki/Merkle_tree)
- bitcoini ametlik lühiinfo (https://en.bitcoin.it/wiki/Protocol_documentation#Merkle_Trees)
- näide bitcoini stackexchangest (https://bitcoin.stackexchange.com/questions/10479/what-is-the-merkle-root)

Siit (https://bitcoin.stackexchange.com/questions/1110/how-do-i-implement-a-merkle-tree) leitud seletus rehkendusele:

In each iteration, you concatenate two subsequent hashes of the previous level, and double-sha256 them. If there is an odd number of hashes, concatenate the last element with itself. This gives you a new list of hashes. Repeat, and stop when one hash remains. This is the merkle root.

Assume you have tx hashes Ha1,Ha2,Ha3,Ha4,Ha5,Ha6,Ha7

- First iteration: Hb1=Hash(Ha1|Ha2), Hb2=Hash(Ha3|Ha4), Hb3=Hash(Ha5|Ha6), Hb4=Hash(Ha7|Ha7)
- Second iteration: Hc1=Hash(Hb1|Hb2), Hc2=Hash(Hb3|Hb4)
- Third iteration: Hd1=Hash(Hc1|Hc2) => Merkle root

NB! Bitcoinis on kombeks arvutada topelt-hashe a la hash(hash(data)) aga meil ei ole mingit põhjust seda teha. Soovitan kasutada lihtsalt ühekordset hashi.

Bloki kaevandamine

Nüüd on sul vaja leida bloki oma hash, kusjuures sobivad ainult sellised hashid, kus on ees N nulli. Katseta, et mitu nulli peaks ees olema, et saaksid ühe sobiva hashi ca viie sekundi jooksul (päris bitcoinis on N-e niipalju, et keskmiselt tuleks üks blokk kümnes minutis kogu võrgu kohta). See nullide nõue on kasutusel ainult selleks, et blokke tuleks harvemini, n.ö. pidur blokkide kiirele genereerimisele.

Selleks võta kogu oma bloki sisu a la

```
{
 "nr": bloki_number_ehk_eelmise_bloki_nr+1,
 "previous_hash": ahelas_eelmise_bloki_hash,
 "timestamp":  "2012-04-21T18:25:43-05:00",
 "nonce": "asasas11",
 "hash": "000000sasas0a0s0111",
 "creator": minu_avalik_võti,
 "merkle_root": "asasasas1w111",
 "count": number_of_transactions,
 "transactions": [{...}, ..., {....}]
}
```

aga **ilma** "nonce" ja "hash" väljadeta ja muuda ta stringiks.

Nüüd tee läbi tsükkel:

- leia lühike random string "nonce" N ja pane see eelnevalt leitud blokistringi S lõppu juurde, saad stringi X.
- arvuta saadud stringi X hash.
- kui saadud hash algab N nulliga, kõik ok ja lõpeta töö.
- vastasel korral mine tsükli esimesele reale tagasi.

Kui sobiv hash leitud, lisa blokile leitud sisuga "nonce" väli ja "hash" väli.

Mitme paralleelbloki korral õige valimine

Sul on üldjuhul suur hulk blokke ahelas. Kui mõni uuematest blokkidest sama "nr" väljaga on topelt, on sul hargnev blokiahel. Siis vali:

- Kõige pikem ahel (kõige suurema numbriga blokk sobib)

- Sama pikkusega ahela puhul vali see, kus
  - harus on rohkem transaktsioone
  - kui see ka sama, vali näiteks see, kus viimane timestamp uuem
  - kui see ka sama, tee näiteks lihtsalt hashide stringivõrdlus ja vali stringijärjekorras väiksema hashiga blokk

Loodud bloki automaatselt teistele edasisaatmine

Kui sul blokk valmis, siis saada ta lihtsalt teistele programmidele edasi. Nemad saadavad seda omakorda edasi ja loodetavasti jõuab see blokk varsti kõigi hetkel võrgus töötavate ja korraliku ühendusega programmideni.

Teine variant: videostriimingu näiterakendus

Sinu rakendus peab suutma võtta vastu brauserist või mobiilist käivitatud videoülesvõtet ja

- Salvestama seda serveris olevasse faili, mida saab pärast läbi brauseri tagantjärgi vaadata. Veebis peab olema näha list kogunenud failidest ja neid peab saama brauseri kaudu vaadata ja failina downloadida.
- Võimaldama läbi teises masinas oleva brauseri seda videoülesvõtet jooksvalt vaadata. Sinu lehel peab olema link või url, millele vajutades hakkad nägema teisest masinast jooksvalt sissetulevat videot.

Arvesta, et erinevalt fotode saatmisest on videostriiming keeruline teema.

Seega tuleks ülesanne lahendada ilma täiendavate keerukusteta: näiteks autentimine, videokvaliteedi valik, kõigi brauseritega kompatiiblus jms tasuks kõrvale jätta. Ei ole vaja suuta salvestada paralleelselt mitut videot. Andmebaasi abil salvestuste manageerimist või suuremate võimalustega UI-d ei ole samuti vaja teha.

Mis on kasulikud alguspunktid:

- Brauseri jaoks on olemas väga hea https://webrtc.org/ süsteem. Loe alustuseks veidi ka wikist (https://en.wikipedia.org/wiki/WebRTC) ja vaata demosid siit (https://webrtc.github.io/samples/), näiteks:
  - https://webrtc.github.io/samples/src/content/getusermedia/record/
  - https://simpl.info/getusermedia/
  - apprtc demo (https://apprtc.appspot.com/) ja selle parameetreid (https://apprtc.appspot.com/params.html)

- Serveri jaoks (et kuidas vastu võtta, faili salvestada ja jooksvalt edasi saata) on kõige kindlam valik https://gstreamer.freedesktop.org Jällegi, loe alustuseks veidi wikist (https://en.wikipedia.org/wiki/GStreamer)

- Vaata kindlasti ka https://opensource.com/article/19/1/gstreamer

Kui sa faili salvestamist või serverist edasisaatmist päris OK käima ei saa, saad praksi ikkagi arvestatud: aga vähemalt üks nendest tuleks käima saada.

- Selle lehekülje viimane muutmine: 15:34, 20. märts 2019.
-