Implementacija kripto valute u programskom jeziku C++

Jukić, Martina-Magdalena

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

IMPLEMENTACIJA KRIPTO VALUTE U PROGRAMSKOM JEZIKU C++

Diplomski rad

Martina-Magdalena Jukić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 15.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Martina-Magdalena Jukić
Studij, smjer:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. Pristupnika, godina upisa:	D-1367, 07.10.2021.
OIB studenta:	39227683731
Mentor:	prof. dr. sc. Marijan Herceg
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Josip Job
Član Povjerenstva 1:	prof. dr. sc. Marijan Herceg
Član Povjerenstva 2:	doc. dr. sc. Denis Vranješ
Naslov diplomskog rada:	Implementacija kripto valute u programskom jeziku C++
Znanstvena grana diplomskog rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Tehnologija ulančanih blokova (engl. blockchain) predstavlja distribuiranu bazu podataka, koja se sastoji od niza podatkovnih blokova povezanih u jednosmjerni lanac u kojem svaki blok ovisi o vrijednosti prethodnog bloka. Kako bi sustavi zasnovani na javnim ulančanim blokovima bili sigurni, koristi se konsenzusni mehanizam kao što je dokaz rada (engl. proof of works), u kojem svaki entitet distribuirane bazi podataka mora uložiti određeni rad kako bi se spriječila zlonamjerna uporaba. Rad se očituje u kriptiranju blokova kako bi se dobio niz znakova određene strukture, a nagrada za obavljeni rad dobiva se u obliku kriptovalute (npr. Bitcoin). U okviru ovog diplomskog rada potrebno je u programskom jeziku C++ kreirati novu kriptovalutu zasnovanu na
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 26.09.2023.

Ime i prezime studenta:	Martina-Magdalena Jukić
Studij:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1367, 07.10.2021.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Implementacija kripto valute u programskom jeziku C++**

izrađen pod vodstvom mentora prof. dr. sc. Marijan Herceg

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

2. PREGLED TEHNOLOGIJA NA KOJIMA SE ZASNIVAJU KRPTO VALUTE 4

2.1. Blockchain 4

2.2. Proof-of-work vs proof-of-stake 6

2.3. Digitalni novčanik 8

2.4. Bitcoin 8

2.5. Legalnost i problematika 12

2.6. Utjecaj na okoliš 12

3. RAZVOJ VLASTITE KRIPTO VALUTE 14

3.1. Priprema okruženja i pokretanje Bitcoin-a 14

3.2. Razvoj FERITcoin-a 15

4. TESTIRANJE 20

5. ZAKLJUČAK 26

LITERATURA 27

SAŽETAK 29

CRYPTOCURRENCY IMPLEMENTATION IN C++ PROGRAMMING LANGUAGE 30

ŽIVOTOPIS 31

PRILOZI 32

1. UVOD

„Kripto valuta je digitalna valuta osmišljena da radi kao sredstvo razmjene novčanih sredstava putem računalne mreže“ [1]. Kripto valute se ne oslanjaju ni na kakva centralna tijela poput vlade ili banke, zbog čega su nestabilne i neregulirane. Temelj im je decentralizirani sustav koji provjerava imaju li strane koje sudjeluju u transakciji dostupna sredstva za transakciju, čime se eliminira potreba za tradicionalnim posrednicima poput banaka. Zapisi o kripto valutama u vlasništvu pojedinca se pohranjuju u digitalnu knjigu, odnosno u bazu podataka koja koristi kriptografiju za osiguranje transakcijskih zapisa, kontrolu stvaranja novih i provjeru prijenosa vlasništva nad novčićima (engl. *coin*). Novčić predstavlja jedinicu kripto valute koja se nalazi na *blckchain*-u. Za rukovanje transakcijama najčešće se koristi tehnologija ulančanih blokova (engl. *blockchain*). Većina kripto valuta prilikom transakcija koristi mehanizme konsenzusa, među kojima su najpoznatiji dokaz o radu (engl. *proof-of-work*) i dokaz o udjelu (engl. *proof-of-stake*), pri čemu vrijeme trajanja transakcije i potrošnja energije uvelike ovisi o odabranom mehanizmu.

Najpoznatija kripto valuta je Bitcoin koja drži oko 50% tržišnog udjela. Pored Bitcoina (BTC), neke od poznatijih kripto valuta su Ethereum (ETH), Binance Coin (BNB), Cardano (ADA), Ripple (XRP), Polkadot (DOT), Dogetti (DETI), Dogecoin (DOGE), Lietcoin (LTC) itd. Većina ovih kripto valuta razvijena je koristeći Bitcoin kao osnovu, zbog čega se često sve kripto valute, koje nisu Bitcoin, kumulativno nazivaju *altcoins* – alternativni novčići.

Razvoj kripto valuta počinje 1983.godine kada je američki kriptograf David Chaum razvio prvi tip kriptografskog električnog novca, koji je nazvao echash. Nešto kasnije, 1955., echash je implementirao u Digicash, koji se smatra prvim oblikom kriptografskog elektroničkog plaćanja. Digicash je radio na osnovu korisničkog softvera koji je umanjivao stanje računa u banci, i kreirao specifične šifrirane ključeve prije nego što se novac mogao poslati primatelju. Šifriranje je onemogućavalo trećoj, neovlaštenoj strani, da uđe u trag digitalnoj transakciji.

1996.godine, National Security Agency, američka obavještajna agencija koja je odgovorna za globalno praćenje, prikupljanje i obradu informacija i podataka, objavila je rad pod imenom „Kako napraviti kovnicu: kriptografija anonimnog elektroničkog novca“, koja opisuje sustav kripto valuta. Rad je podijeljen putem MIT-eve e-mail liste, a godinu kasnije i službeno objavljen u The American Law Review.

Wei Dai, kineski inženjer računarstva koji je razvio Crypto++ biblioteku, 1998.godine opisao je b-money, anonimni i distribuirani elektronički novčani sustav. Nedugo nakon toga, Nick Szabo,

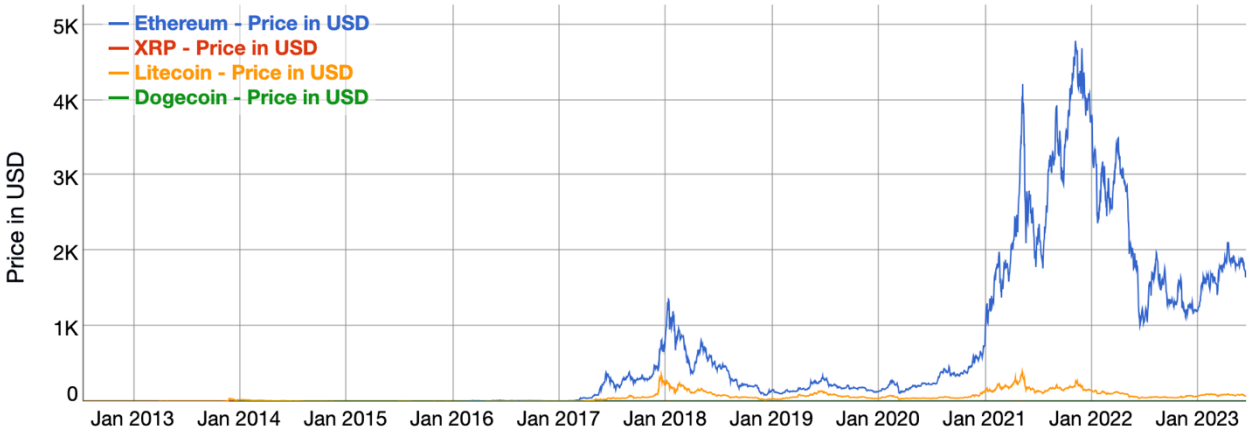
pravni znanstvenik i kriptograf, opisao je bit gold, koji je, kao i većina kripto valuta koje su slijedile, poput Bitcoina, od korisnika zahtijevao da se izvrši *proof-of-work* kako bi se blok validirao.

Prvi decentralizirani *blockchain* je konceptualiziran od strane „Satoshi Nakamoto“ 2008.godine, gdje je, do tad postojeći *blockchain* protokol, poboljšao dodavanjem korištenja *hash*-ova kako bi se blokovi vremenski označili bez potrebe da oni budu potpisani od pouzdane strane, i uvođenjem parametra težine bloka za stabilizaciju brzine dodavanja blokova u lanac. Dodavanjem vremenske oznake i *hash*-iranjem, korisnika se sprječava da troši više novca nego što ima, tj. onemogućuje se da se novac pojavi niodkud, zbog čega nije potrebna pouzdana strana (banka, vlada). Iduće godine, u siječnju 2009., Nakamoto-u je ova tehnologija poslužila kao osnova za implementiranje kripto valute Bitcoin, gdje se *blockchain* koristi kao javna digitalna knjiga za sve transakcije u mreži. Bitcoin koristi SHA-256 (engl. *Secure Hash Algorithm*) kao kriptografsku *hash* funkciju i *proof-of-work* shemu.

U travnju 2011.godine, kreiran je Namecoin kao pokušaj formiranja decentraliziranog DNS-a (engl. *Domain Name System*). Litecoin je objavljen u listopadu 2011.godine i koristi Scrypt kao *hash* funkciju. Peercoin koristi hibrid *proof-of-work* i *proof-of-stake* metoda, kreiran je u kolovozu 2012.godine. Na slici 1.1. pokazana je vremenska crta razvoja kripto valuta.

Slika 1.1. Vremenska crta pojave kripto valuta [2]

Kripto valute su prošle kroz nekoliko razdoblja rasta i pada tržišne vrijednosti, a prvi veći pad se dogodio krajem 2021.godine, kada je Kina, najveće tržište kripto valuta poslije Amerike, proglasila transakcije kripto valutama ilegalnima. Na slici 1.2. pokazan je pad i rast tržišne vrijednosti najpoznatijih *altcoin*-ova.



Slika 1.2. Tržišne vrijednosti najpoznatijih altcoin-ova [3]

Jordan Kelly, osnivač Robocoin-a, pokrenuo je prvi Bitcoin bankomat u Austinu, Texas, 2014.godine. Bankomat ima skenere za očitavanje identifikacijskih dokumenata, poput vozačke dozvole ili putovnice za potvrdu identiteta korisnika. Trenutno je Bitcoin bankomat popularan u Europi, a tako i u Osijeku kao što je pokazano na slici 1.3.

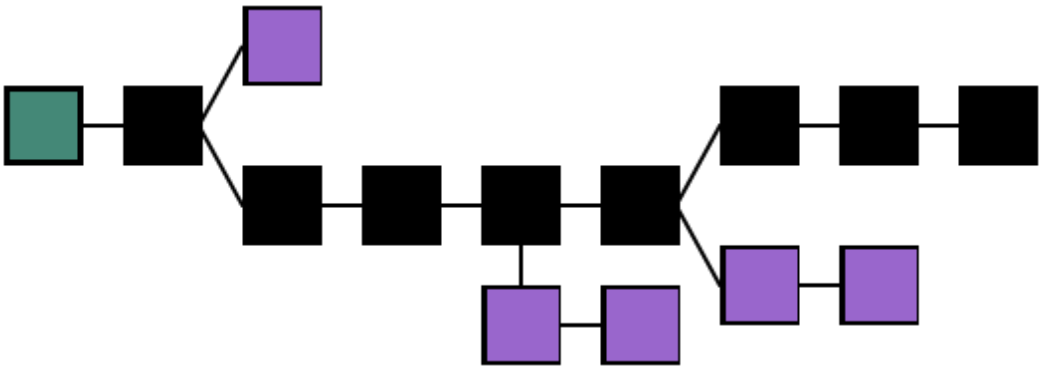
Slika 1.3. Bitcoin bankomat u Osijeku (Portanova)

2. PREGLED TEHNOLOGIJA NA KOJIMA SE ZASNIVAJU KRPTO VALUTE

Temeljni tehnološki sustav na kojem su baziranje kripto valute kreirao je „Satoshi Nakamoto“, tj. kripto valute su temeljene na *blockchain* tehnologiji za spremanje transakcija. Za validaciju transakcija najčešće se koriste *proof-of-work* (PoW) ili *proof-of-stake* (PoS) metode. Pored njih, poznate su i praktična bizantska tolerancija na pogreške (engl. *practical byzantine fault tolerance*), delegirani dokaz o udjelu (engl. *delegated proof of stake*), dokaz opekline (engl. *proof of burn*), dokaz o kapacitetu (engl. *proof of capacity*), dokaz o proteklom vremenu (engl. *proof of elapsed time*), itd.

2.1. Blockchain

„*Blockchain* je distribuirana digitalna knjiga s rastućom listom zapisa (blokova) koji su međusobno povezani na siguran način pomoću kriptografskih *hash*-ova“ [4]. Svaki blok sadrži kriptografski *hash* prethodnog bloka, vremensku oznaku i transakcijske podatke (koji se uglavnom prikazuju kao *Hash* stablo ili, po izumitelju, Merkle stablo, gdje se podatkovni čvorovi prikazuju kao listovi). S obzirom da svaki blok sadrži podatke o prethodnom bloku, oni zajedno tvore lanac (engl. *chain*), zbog čega su *blockchain* transakcije ireverzibilne, jer, kada su jednom transakcije spremljene, nemoguće ih je naknadno mijenjati a da se ne mijenjaju svi prethodni blokovi. Inicijalni blok (Blok 0) naziva se *genesis block* koji je ključan za svaku kripto valutu, kako bi se osigurao integritet bloka i podataka sadržanih u njemu, ovaj blok se uglavnom digitalno potpisuje.



Slika 2.1. Blockchain struktura [5]

Prema slici 2.1., zeleni blok predstavlja *genesis* blok, crni blokovi predstavljaju glavni lanac, a ljubičasti blokovi, koji se nalaze van glavnog lanca, nazivaju se siroče (engl. *orphan*) blokovi. Kod kripto valuta, *orphan* blokovi predstavljaju blokove koji su validirani rudarenjem, ali odbačeni.

Takvi blokovi nastaju kada dva rudara gotovo istovremeno dodaju blok u lanac, čime se lanac dijeli na dvije strane, gdje je jednu potrebno odbaciti.

Upravljanje *blockchain*-om se uglavnom izvodi pomoću jednak-prema-jednakome (engl. *peer-to-peer*) računalne mreže, gdje se čvorovi pridržavaju algoritma konsenzusa (PoW ili PoS) za dodavanje i provjeru valjanosti novih blokova transakcija. Iako zapisi *blockchain*-a nisu nepromjenjivi, da bi se mijenjao jedan blok moraju se promijeniti i svi njegovi prethodni blokovi, što zahtijeva veliku računalnu moć, zbog čega se mogu smatrati sigurnima. *Blockchain* se smatra sigurnim po dizajnu i primjer je distribuiranog računalnog sustava s visokom bizantskom[1] tolerancijom na pogreške.

Implementacija *blockchain*-a unutar Bitcoin-a riješila je problem *double-spending*-a[2] bez potrebe za pouzdanim tijelom ili centralnim poslužiteljem. *Blockchain* se može smatrati vrstom sustava plaćanja.

Može se reći kako se *blockchain* sastoji od više slojeva:

- infrastrukture (engl. *hardware*)
- umrežavanja (otkrivanje čvorova, informacijska propagacija i verifikacija)
- metode konsenzusa (PoW/PoS)
- podataka (blokovi, transakcije)
- aplikacije (pametni ugovori, decentralizirane aplikacije)

Postoje barem četiri vrste *blockchain* mreža:

- javni *blockchain* – nema nikakvih ograničenja pristupa, bilo tko s internetskom vezom može slati transakcije i postati validator (izvršavati neku metodu konsenzusa), Bitcoin i Ethereum koriste javni *blockchain*,
- privatni *blockchain* – zahtijeva dozvolu pristupa, mreži se ne može pristupiti bez poziva administratora mreže, preporučuju se za poslovnu upotrebu,
- hibridni *blockchain* – kombinacija centraliziranih i decentraliziranih značajki,
- konzorcijski *blockchain* – objedinjuje elemente privatnog i javnog *blockchain*-a, grupa organizacija zajedno kreiraju i upravljaju *blockchain*-om umjesto jednog entiteta.

[1] Bizantska pogreška je stanje računalnog sustava, posebno distribuiranih računalnih sustava, gdje komponente mogu zakazati i postoje nesavršene informacije o tome je li komponenta otkazala[6].

[2] Dvostruka potrošnja – trošenje istog novca više puta kao posljedica mijenjanja informacija o transakcijama što se implementacijom *blockchain*-a onemogućilo

Pored kripto valuta, *blockchain* tehnologija se može koristiti za stvaranje trajnog, javnog, transparentnog sustava knjiga za prikupljanje podataka o prodaji, praćenju digitalne upotrebe i plaćanja kreatorima sadržaja, kao što su npr. glazbenici. Koristi se i u *peer-to-peer* trgovini energijom, kao i za otkrivanje krivotvorina povezivanjem jedinstvenih identifikatora s proizvodima, dokumentima i pošiljkama.

Uz pojavu sve većeg broja *blockchain* sustava (općenito, ali i vezano konkretno za kripto valute), interoperabilnost postaje tema od velike važnosti. Cilj je podržati prijenos sredstava s jednog *blockchain* sustava na drugi. Već postoji nekoliko dostupnih rješenja za interoperabilnost *blockchain*-a, a mogu se klasificirati u tri kategorije: pristupi interoperabilnosti kripto valuta, *blockchain* motori i *blockchain* konektori.

2.2. Proof-of-work vs proof-of-stake

Mehanizmi konsenzusa pomažu *blockchain*-u u sinkronizaciji podataka, validaciji informacija i procesiranju transakcija. Najpoznatije metode konsenzusa su PoW i PoS.

PoW zahtijeva značajnu računalnu moć od mreže uređaja. Kripto valutama je koncept prilagodio Hal Finney kroz ideju „dokaza o radu za višekratnu upotrebu“ koristeći 160-bitni sigurni *hash* algoritam (SHA) [7]. PoW je decentralizirani mehanizam konsenzusa koji od članova mreže zahtijeva pronalaženje šifriranog heksadecimalnog broja s određenim karakteristikama. Kada se kreira blok koji nosi transakcijske informacije, on se kriptira *hash* algoritmom. U slučaju Bitcoin-a, rezultat je 64 znamenkasti kriptirani heksadecimalni broj koji je moguće generirati u nekoliko milisekundi ako je u pitanju velika količina podataka. Uloga validatora, koji se u ovom slučaju zovu rudari (engl. *miners*), je da „pogode“ taj heksadecimalni broj, što je vremenski i procesno jako zahtjevan posao. Heksadecimalni broj je „pogođen“ ukoliko je *hash* manji od trenutnog mrežnog cilja[3], tj. tada je transakcija validirana. Taj proces naziva se rudarenje (engl. *mining*) zbog čega se i PoW često poistovjećuje s pojmom rudarenje podataka. Rudar koji riješi *hash* dobiva nagradu za obavljeni posao u obliku novčića, npr. nekoliko Bitcoin-a, zbog čega su osobe koje rudare u utrci za ostvarivanjem što veće računalne snage.

PoS je kreiran kao alternativa za PoW. Validatori se odabiru na osnovu dostupnih i uloženih novčića, kako bi mogli zaraditi na transakcijskim naknadama. Za razliku od PoW gdje postoji natjecateljstvo za rješavanjem *hash*-eva, u PoS se validatori odabiru nasumično, ali vjerojatnosti

[3] Mrežni cilj je pokazatelj poteškoće rudarenja pretvoren u heksadecimalni broj.

za odabir ovise o udjelu novčića koji pojedinac posjeduje [8]. Primjer PoS je *sharding*[4] koji koristi Ethereum, gdje validator potvrđuje transakcije i dodaje ih u *shard* blok, koji zahtijeva minimalno 128 validatora da bi se kreiralo glasačko povjerenstvo. Kada je *shard* validiran i blok kreiran, dvije trećine validatora se moraju složiti kako je transakcija validna čime se blok zatvara. Kako bi netko postao validator, mora posjedovati bar 32 ETH.

Usko povezano s metodama konsenzusa je vrijeme bloka (engl. *blocktime*), tj. vrijeme potrebno za generiranje novog bloka u *blockchain*-u što uključuje i validaciju. Kada je riječ o kripto valutama, to se odnosi na vrijeme trajanja transakcije, što je kod Etheruma 12 sekundi, a kod Bitcoin-a oko 10 minuta. Zbog toga se kripto valute bazirane na PoS smatraju ekološki održivim opcijama, zbog čega je i Ethereum u rujnu 2022.godine s PoW prešao na PoS čime je smanjena potrošnja energije i emisija ugljičnog dioksida za 99,9%. U tablici 2.1. prikazane su osnove razlike između PoW i PoS metodi konsenusa.

Tablica 2.1. Osnovne razlike PoW vs PoS

proof-of-work	**proof-of-stake**
Validaciju radi mreža „rudara“	Validaciju provode sudionici koji nude ETH kao kolateral
Rudari moraju posjedovati opremu i trošiti energiju kako bi sudjelovali u procesu validacije	Validatori moraju posjedovati novčiće kako bi sudjelovali u procesu validacije
Energetski neučinkovit	Energetski učinkovit
Čvrsta sigurnost zbog skupih zahtijeva	Sigurnost kroz kontrolu zajednice
Rudari dobivaju Bitcoin-e kao nagradu i za transakcijske naknade	Ether se plaća samo za transakcijske naknade

I PoW i PoS imaju zajedničku prijetnju – hakerski napad pod nazivom „51% attack“. Kad je u pitanju PoW, to je slučaj u kojem entitet kontrolira preko 50% rudara u mreži i koristi tu većinu kako bi promijenio *blockchain*, a kod PoS kada pojedinac ili grupa ljudi posjeduju preko 50% novčića koji su u optjecaju. I u jednom i u drugom slučaju je to malo vjerojatno jer takav napad zahtijeva velike troškove.

[4] Engl. shard – manji dio nečeg većeg. Sharding nastoji blockchain podijeliti na više manjih dijelova od kojih svaki sadrži podskup transakcijskih podataka čime se omogućuje paralelno obrađivanje transakcija što kao posljedicu ima poboljšanje latencije i povećanje skalabilnosti mreže[9].

2.3. Digitalni novčanik

Digitalni novčanik (engl. *wallet*) pohranjuje informacije potrebne za transakcije, a odnosi se na posjed novčića koji ima pojedinac. Zbog prirode sustava, novčići su neodvojivi od *blockhain*-a, tako da digitalni novčanik može služiti samo za pohranjivanje digitalne vjerodajnice za posjed novčića i omogućuje pristup novčićima i njihovo trošenje. Bitcoin koristi kriptografiju javnim ključem, gdje se generira par ključeva – javni ključ za šifriranje i privatni ključ za dešifriranje. U najosnovnijem smislu, digitalni novčanik bi bio skup ovih ključeva.

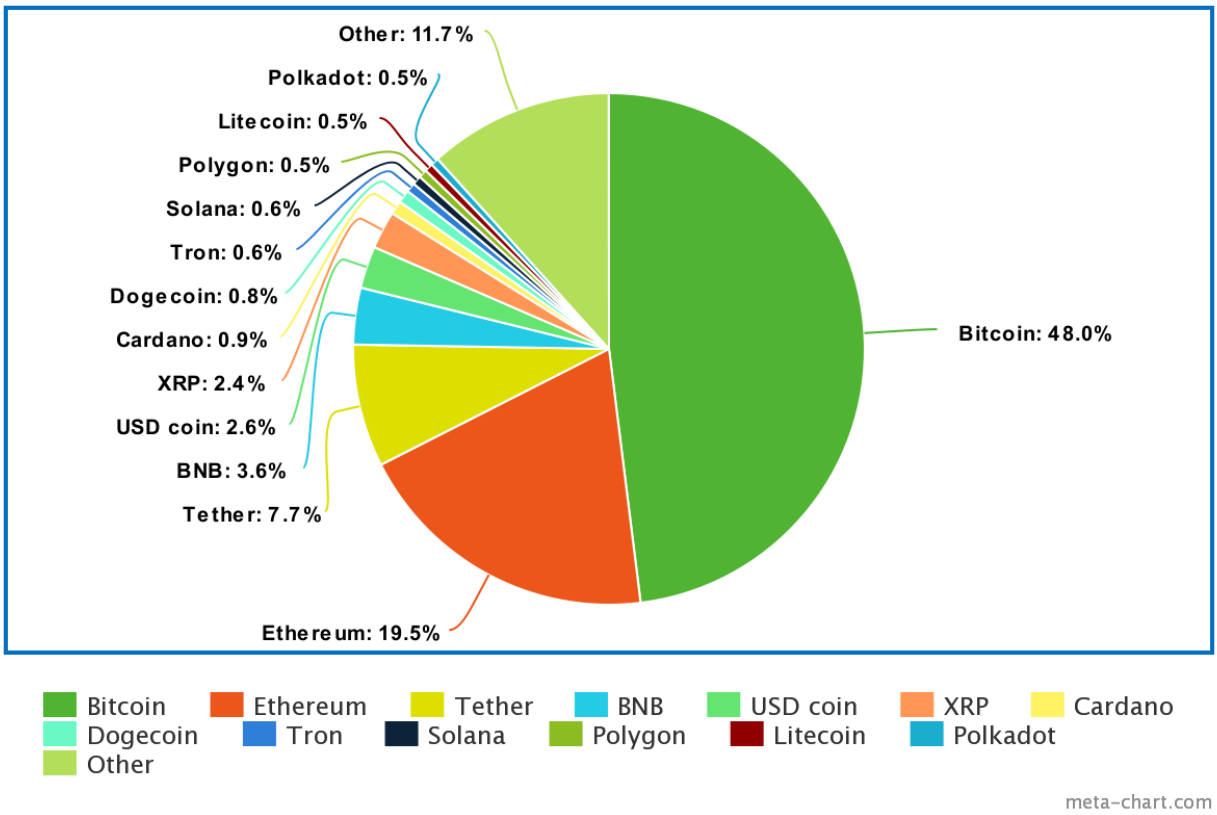
Općenito postoje četiri vrste digitalnih novčanika:

- softverski novčanik (engl. *software wallet*) – program za pohranjivanje ključeva, prvi se pojavio pojavom Bitcoin-a početkom 2009. i po izumitelju se često naziva Satoshi klijent,
- hladni spremnik (engl. *cold storage*) – pošto su digitalni novčanici česta meta hakera, u ovoj metodi su privatni ključevi uvijek *offline*, tj. ključevi se generiraju na uređaju koji nije spojen na internetsku mrežu,
- hardverski novčanik (engl. *hardware wallet*) – računalna periferija koja potpisuje transakcije prema zahtjevu korisnika, uređaj koji sadrži privatne ključeve i potpisuje i šifrira informacije interno i ne dijeli transakcije koje nisu već potpisane, i samim time neizmjenjive, uređaj je zaštićen šifrom,
- papirni novčanik (engl. *paper wallet*) – par ključeva se generira na uređaj bez pristupa internetskoj mreži te se privatni ključ zapisuje ili printa na komad papira te briše s uređaja.

Entitet može posjedovati više novčanika, čime je onemogućeno da se točno odredi kojem pojedincu, grupi ljudi ili entitetu pripada koja količina novčića.

2.4. Bitcoin

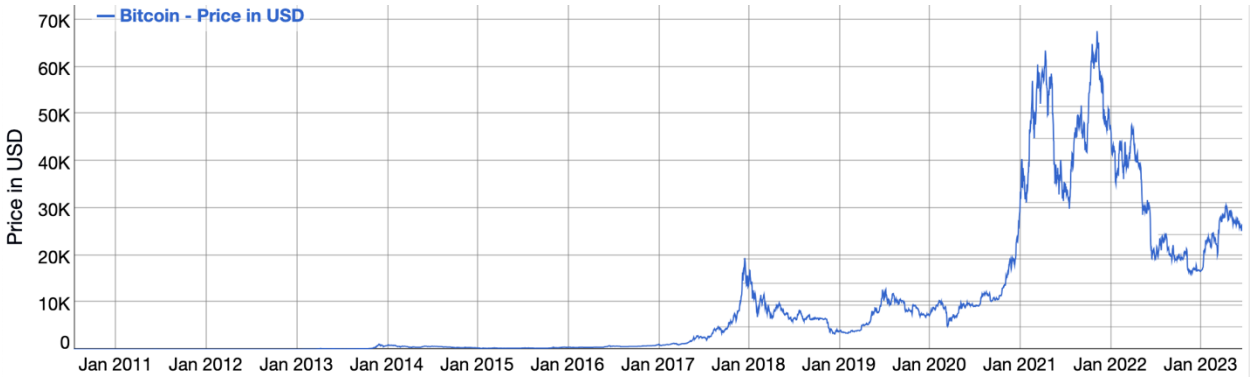
Bitcoin je kripto valuta, virtualna valuta dizajnirana da djeluje kao novac i oblik plaćanja izvan kontrole bilo koje osobe, grupe ili entiteta, čime se uklanja potreba za sudjelovanjem treće strane u financijskim transakcijama. Od pojave Bitcoina 2009.godine, pa do dan danas je to najpopularnija kripto valuta koja čini oko 50% tržišnog udjela. Na slici 2.2. prikazani su tržišni udjeli najpopularnijih kripto valuta.



Slika 2.2. Tržišni udio kripto valuta 17.06.2023. [10]

Bitcoin je *pseudonymous*, što znači da sredstva nisu vezana za entitete iz stvarnoga svijeta već za Bitcoin adrese pomoću kojih se spremaju u *blockchain*. Vlasnici Bitcoin adresa nisu eksplicitno identificirani, ali su sve transakcije na *blockchain*-u javne i na taj način se transakcije mogu povezivati s pojedincima ili tvrtkama putem „idioma upotrebe“. Dobra praksa je za svaku uplatu kreirati novu adresu, tako da jedan novčanik obično posjeduje velik broj različitih adresa. Bitcoin mjenjačnice gdje se Bitcoin trguje za tradicionalne valute moraju zakonski prikupljati osobne podatke. Zbog anonimnosti koju strane u transakciji mogu zadržati, Bitcoin se često može koristiti i u ilegalne svrhe, zbog čega postoji otpor prema kripto valutama.

Najpoznatiji slučaj gdje je se Bitcoin koristio u ilegalne svrhe je Silk Road, stranica na *darknetu*-u gdje su se prodavale droge i koristila pošta kao sredstvo dostavljanja robe. Bitcoin je tu bila glavna valuta pošto se takvim transakcijama nije moglo ući u trag što je osiguravalo anonimnost kupaca i prodavača robe. Najpoznatiji legalni slučaj korištenja Bitcoin-a u srhe poslovanja je kada je Elon Musk u lipnju 2021. godine počeo prihvaćati Bitcoin kao sredstvo plaćanja za kupovinu Tesla automobila, što je uvelike utjecalo na popularnost i vrijednost Bitcoin-a. Na slici 2.3. prikazana je vrijednost Bitcoin-a u odnosu na američki dolar u posljednjih 13 godina.



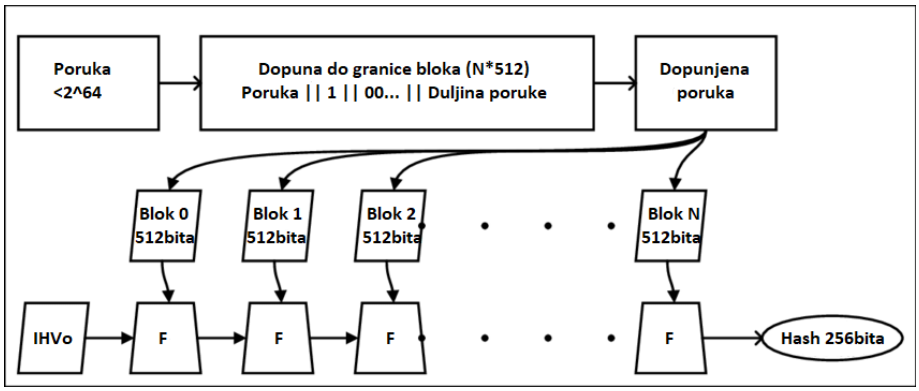
Slika 2.3. Vrijednost Bitcoina u odnosu na USD [3]

Povećanjem broja Bitcoina koji su u optjecaju, povećava se i veličina njegovog *blockchain*-a. U kolovozu 2014.godine, veličina *blockchain*-a je dosegnula 20GB, u siječnju 2015.godine je veličina narasla na 30GB, a od 2016.godine do 2017.godine, veličina *blockchain*-a se poduplala s 50GB na 100GB. Veličina digitalne knjige je prešla 200GB početkom 2020.godine, a trenutno iznosi 489,16GB [11].

Bitcoin koristi PoW metodu konsenzusa koja ima veliku potrošnju energije, zbog čega se rudari koji validiraju transakciju dobro nagrađuju, trenutno ta nagrada iznosi 6,25 BTC, što bi bilo 165.156,25 američkih dolara. Zbog želje za zaradom, više rudara ili grupa rudara istovremeno pokušava validirati blok što rezultira puno većom potrošnjom od one što je potrebna, što ima negativan utjecaj na okoliš.

Hashing algoritam koji koristi Bitcoin je SHA-256. Algoritmi se mogu koristiti u razne svrhe, ali SHA algoritmi se iskorištavaju u svrhe kriptografske sigurnosti. SHA-256 je član SHA-2 kriptografskih *hash* funkcija dizajniranih od strane NSA[5], objavljen 2004.godine. SHA-256 generira gotovo unikatan 256-bitni potpis (32 bajta) za dani tekst koji nije reverzibilan. Ukoliko se samo posjeduje *hash* nemoguće je doći do originalnih podataka. Sigurnost SHA-256 algoritma dijelom je ovisna o njegovoj rezistentnosti na koliziju, koja se ogleda kao vjerojatnost da dva različita podatkovna ulaza daju isti *hash* izlaz [12]. Na slici 2.4. je prikazana generalna shema SHA-256 *hashing* algoritma gdje F predstavlja 64 ponavljanja funkcije runde, a IHV_0 inicijalnu *hash* vrijednost (engl. *Initial Hash Value*). Inicijalne *hash* vrijednosti su konstantne i izvode se iz razlomaka kvadratnih korijena prvih osam prostih brojeva [13].

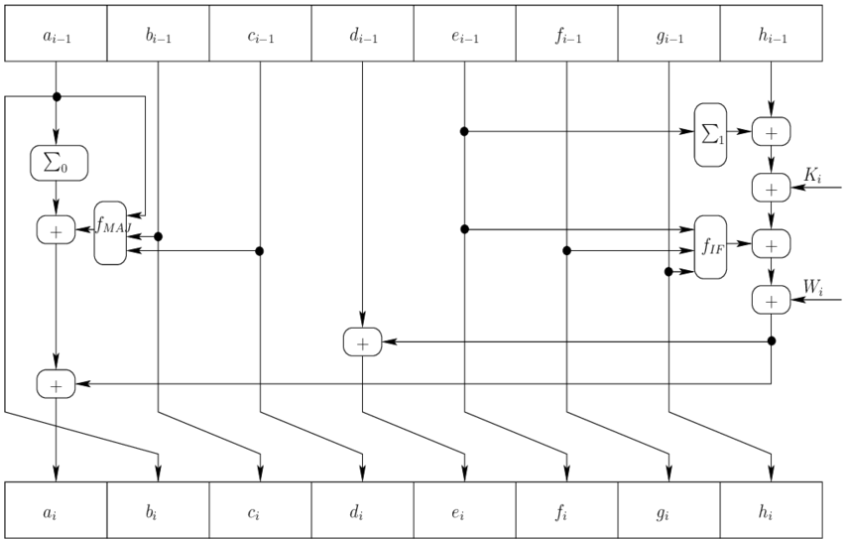
[5] Agencija za nacionalnu sigurnost (engl. *National Security Agency*) je obavještajna agencija na nacionalnoj razini Ministarstva obrane Sjedinjenih Država.



Slika 2.4. Generalna shema SHA-256 *hashing* algoritima [13]

SHA-256 algoritam se može podijeliti na pet dijelova:

- bitovi za dopunu (engl. *padding bits*) – poruci koju je potrebno šifrirati se dodaju bitovi tako da duljina bude točno 64 bita manja od višekratnika broja 512, prvi bit koji se dodaje je 1, a ostali su 0,
- duljina dopune (engl. *padding length*) – dodaju se 64 bita koji predstavljaju duljinu originalne poruke u bitima kako bi se dobio krajnji tekst za obradu,
- inicijalizacija međuspremnika (engl. *buffer initialization*) – inicijalizacija vrijednosti za 8 međuspremnika i 64 različita ključa u nizu (od K[0] do K[63]),
- funkcije kompresije (engl. *compression functions*) – poruka se razdvaja u više blokova po 512 bita pri čemu svaki blok prolazi kroz 64 runde gdje izlaz svakog bloka služi kao ulaz u sljedeći blok. Na slici 2.5. je prikazana funkcija runde koju koristi SHA-256. Vrijednosti K[i] su inicijalizirane unaprijed, a W[i] se računa za svaki blok pojedinačno, ovisno o broju ponavljanja koji se obrađuje [14].



Slika 2.5. Funkcija runde SHA-256 algoritma [15]

2.5. Legalnost i problematika

Iako većina ljudi neovisnost o državi i bankama smatra novom razinom slobode, postoji i dosta kritičara zbog nestabilnosti kripto valuta. Howard Marks, američki investitor, rekao je kako su kripto valute „ništa više od neutemeljenog hira (ili čak piramidalne sheme), temeljene na spremnosti da se pripiše vrijednost nečemu što ima jako malu ili čak nikakvu vrijednost osim one što bi ljudi za to platili“[16]. Pored nestabilnosti, problem stvaraju i anonimnost i manjak regulacija, koje iskorištavaju web kriminalci za utajivanje poreza i pranje novca, zbog čega su neke zemlje uvele zabranu korištenja kripto valuta.

Sjedinjene Američke Države su 2021. u 17 zemalja proglasile zakone i rezolucije za regulaciju kripto valuta. Regulacije su donesene iz razloga što investitori mogu biti ranjivi bez posrednika kada trguju kripto valutama i iz straha kako mnogi novčići mogu biti neregistrirani vrijednosni papiri bez tržišnog nadzora. Donesen je zakon kako sve transakcije iznad 10.000 američkih dolara moraju biti prijavljene IRS-u[6] kako bi se mogao obračunati porez. Način da se stane ilegalnim transakcijama na kraj se još traži.

2021.godine Kina je zabranila financijskim institucijama i tvrtkama da koriste kripto valute kao način plaćanja, a naknadno i totalnu zabranu kripto valuta kako bi se stalo u kraj rudarima u Kini koji su koristili velike količine električne energije koje su generirane iz zagađujućih izvora kao što je ugalj. To je dovelo do oštrog pada vrijednosti Bitcoin-a i prelaska Ethereum-a na PoS. El Salvador je u lipnju 2021.godine objavio kako će usvojiti Bitcoin kao zakonsko sredstvo plaćanja, čime je postao prva zemlja koja je to učinila, nakon čega je slijedila i Kuba u kolovozu 2021.godine.

Brojne humanitarne agencije počele su primati donacije u kripto valutama, među kojima je i UNICEF, ali s poštovanjem protokola koji on ima, tj. da svi donatori prolaze provjere prije nego što se dopusti uplata sredstava. Ukrajinska vlada je 2022. prikupila pomoć u vrijednosti od preko 10.000.000 američkih dolara putem kripto valuta nakon invazije Rusije.

2.6. Utjecaj na okoliš

Rudarenje kripto valuta zahtijeva enormne količine električne energije i dolazi s velikom emisijom ugljičnog dioksida. Procjenjuje se kako su *blockhain*-i s PoW metodom konsenzusa (Bitcoin,

[6] Internal Revenue Service – služba za prihode Sjedinjenih Američkih Država koja je odgovorna za prikupljanje poreza

Ethereum, Litecoin i Monero) u razdoblju od 1.siječnja 2016.godine do 20.lipnja 2017.godine doprinijeli ispuštanju 3-15 milijuna tona ugljičnog dioksida u atmosferu.

Procjenjuje se kako je u 2022. Bitcoin bio zadužen za 0,1% svjetske emisije stakleničkih plinova, CCAF[7] je procijenio kako je Bitcoin potrošio 100 TWh u 2022.godini, što bi odgovaralo potrošnji Egipta. Pored velike potrošnje energije, negativan utjecaj na okoliš ima i činjenica da se dobar dio te energije proizvodi iz ugljena, a i e-otpad koji nastaje zbog kratkog životnog vijeka opreme koja se koristi za rudarenje Bitcoin-a.

Zbog zabrinutosti oko potrošnje energije i drugih čimbenika, Kina je u lipnju 2021.godine zabranila korištenje Bitcoin-a, zbog čega SAD postaje vodeći svjetski lider u industriji rudarenja. Među najvećim postrojenjima za rudarenje spominju se postrojenje u Daltonu, Georgia, koje troši količinu energije kao 97.000 kućanstava u njegovoj blizini, i Riot Platforms u Rockedaleu, Teksas, koje troši količinu energije kao 300.000 kućanstava.

Potrošnja PoS *blockchain*-ova je na razini ekvivalentnoj stambenim naseljima, za razliku od PoW koji troše kao zemlje srednje veličine. Zbog toga se PoS kripto valute smatraju ekološki prihvatljivim rješenjem gdje je The Times definirao šest najekološkijih kripto valuta : Chia, IOTA, Cardano, Nano, Solarcion i Bitgreen. Studija o šest najvećih PoS kritpto valutama pokazala je kako Cardano ima najmanju potrošnju energije po čvoru, Polkadot ukupno najmanju potrošnju energije i Solana najmanju potrošnju energije po transakciji. Što se tiče godišnje potrošnje, Polkadot troši energije kao 7 prosječnih američkih kućanstava, Cardano troši prosječno energije kao 57 kućanstava, a Solana kao 200 kućanstava. Kripto valute temeljene na PoS metodi konsenzusa troše 0,001% električne energije Bitcoin mreže. Kako bi smanjio svoj negativni utjecaj na okoliš, najveći altcoin, Etehereum, je u rujnu 2022. prešao s PoW na PoS.

[7] Cambridge Centre for Alternative Finance – istraživački institut koji se fokusira na financijske kanale i instrumente koji se pojavljuju izvan tradicionalnih financijskih ekosustava.

3. RAZVOJ VLASTITE KRIPTO VALUTE

Kao i većina *altcoin*-ova, i razvoj vlastite kripto valute u ovom diplomskom radu će biti zasnovan na konceptu Bitcoin-a. Koristit će se verzija 0.18.1 Bitcoin Core-a iz razloga što je ona najbolje dokumentirana i najviše primjera za razvoj vlastite kripto valute je rađen upravo na toj verziji.

3.1. Priprema okruženja i pokretanje Bitcoin-a

Za pokretanje izvršnih datoteka potreban je operacijski sustav nalik OS X-u ili Unix-u, zbog čega je instaliran Ubuntu18.04 u virtualnoj mašini. Kako bi se mogao izgraditi (engl. *build*) kod, potrebno je instalirati pakete s potrebnim datotekama (engl. *dependencies*) kako je navedeno u službenoj dokumentaciji za Bitcoin [17]. Nakon instalacije svih *dependencies* potrebno je dohvatiti kod za Bitcoin Core sa službene Bitcoin stranice na GitHub-u pomoću *git* naredbi. Kada se završi skidanje koda, potrebno je pomoću komandnog sučelja navigirati u novi direktorij te izgraditi kod prema uputama sa službene dokumentacije.

Pokretanje naredbi `./autogen.sh` i `./configure` je prošlo uspješno, ali pokretanjem naredbe `make` su dobivene greške vezano za pojedine biblioteke koje se uključuju, što je bilo potrebno riješiti. Nakon što su riješeni problemi i ponovno pokrenut `make` uspješno su generirane tri izvršne datoteke: `bitcoind`, `bitcoin-cli` i `bitcoin-qt`. `bitcoind` uspostavlja *blockchain* i pokreće čitav sustav te omogućuje daljnje izvođenje naredbi poput rudarenja i izvršavanja transakcija, pomoću `bitcoin-cli` moguće je provjeriti stanje u kojem se nalazi *blockchain*, mreža, digitalni novčanik i slično, kao i zaustavljanje bitcoin servera pokrenutog pomoću `bitcoind`, a `bitcoin-qt` pokreće digitalni novčanik, tj. korisničko sučelje u kojem je moguće provjeriti koji iznos je dostupan i izvršavanje transakcija. Pokretanjem naredbe `./bitcoind -printtoconsole` u `bitcoin/src/` direktoriju uspostavlja se *blockchain*, što je proces koji traje nešto više od sat vremena, tijekom kojeg i nakon kojeg se pomoću naredbe `./bitcoin-cli getblockchaininfo` može provjeriti stadij do kojeg je došlo ili njegova uspješnost. Nakon uspješnog uspostavljanja *blockchain*-a dobije se ispis pokazan u prilogu P.3.1. Iz priloga se može očitati kako se radi o `main` lancu (postoje još i `test` i `regtest`), blokova i zaglavlja ima nula, što znači da u lancu postoji samo *genesis* blok koji je prikazan kao `bestblockhash`, težina (engl. *difficulty*) je 1, ali kako raste veličina lanca raste i težina generiranja novih *hash*-ova. `mediantime` odgovara vremenskoj oznaci (engl. *timestamp*) dobivenoj prilikom generiranja *genesis* bloka, `initialblockdownload` postavljen na *true* označuje da se već postojeći *blockchain* neće ponovno generirati nego da će se skinuti s interneta. `size_on_disk` pokazuje veličinu memorije koju zauzima *blockchain*, u ovom

slučaju je to veličina *genesis* bloka koja iznosi 293kB, te je moguće vidjeti kako nema nikakvih upozorenja (engl. *warnings*). Nakon potvrde da je sve uspješno instalirano i da se Bitcoin može izgraditi i pokrenuti, može se krenuti s razvojem vlastite kripto valute.

3.2. Razvoj FERITcoin-a

Kako bi se krenulo s razvojem vlastite kripto valute, prvotno je potrebno promijeniti sva imena gdje se spominje u Bitcoin, kako u nazivima datoteka, tako i u datotekama, pri čemu se mora paziti da su sva imena dobro promijenjena inače će se vjerojatno pojaviti problem u `Makefile`-ovima zbog čega se projekt neće moći izgraditi. Imena unutar datoteka su promijenjena sljedećim naredbama:

```
find ./ -type f -readable -writable -exec sed -i „s/Bitcoin/Feritcoin/g“ {} \;
find ./ -type f -readable -writable -exec sed -i „s/bitcoin/feritcoin/g“ {} \;
find ./ -type f -readable -writable -exec sed -i „s/bitcoind/feritcoind/g“ {} \;
find ./ -type f -readable -writable -exec sed -i „s/BTC/FRT/g“ {} \;
```

Imena datoteka su promijenjena naredbom `rename` pri čemu je se moralo ulaziti u svaku mapu zasebno kako bi promjene bile primijenjene na čitav bitcoin direktorij:

```
rename 's/bitcoin/feritcoin/g' *
```

Nakon promijene svih imena promijenjeno je ime direktorija iz `bitcoin` u `FERITcoin` [18] te su `Makefile`-u napravljene potrebne promjene. Projekt je potrebno rekonfigurirati pokretanjem naredbe `./configure` nakon čega se projekt izgradi (naredba `make`) i provjerava se uspješnost pokretanjem nove naredbe `./feritcoind` i `./feritcoin-cli getblockchaininfo`.

Pored imena, bitno je promijeniti još nekoliko osnovnih stvari kako bi se stvoreni projekt mogao nazivati vlastitom kripto valutom. Većina kripto valuta koristi ili PoW ili PoS kao metodu konsenzusa, tako da će konsenzus ostati nepromijenjen (PoW), ali svaka kripto valuta ima različit *genesis* blok i često implementiraju svoje *hashing* algoritme. Tako će se i u ovom diplomskom radu mijenjati *genesis* blok, što utječe na čitav *blockchain*, kao i *hashing* algoritam.

„Satoshi Nakamoto“ je za stvaranje *genesis* bloka koristio tekst „The Times 03/Jan/2009 Chancellor on brink of second bailout for banks“ s naslovnice novina, čime je to postala tradicija pri kreiranju kripto valuta. Za kreiranje *genesis* bloka za ovaj diplomski rad korišten je naslov iz Večernjeg lista: „Vecernji list 22/June/2023 Tvrtka OceanGate objavila: Vjerujemo da su svi putnici mrtvi“, a za generiranje bloka korišten je alat GenesisH0[19] dostupan na GitHub-u. Nakon

pet sati uspješno je generiran blok *hash*, korijen Merkle stabla, *nonce* (engl. *number only used once*) i vremenska oznaka koje je potrebno zamijeniti u kodu (u datoteci `chainparams.cpp` u `FERITcoin/src/`). Kako bi se smanjilo vrijeme potrebno za generiranje *genesis* bloka, može se smanjiti „težina“, tj. može se zadati dodatan parametar sa zastavicom -b prilikom korištenja GenesisH0 alata i kao treći parametar (`nBits`) u funkciju `CreateGenesisBlock()` u `chainparams.cpp`. Parametar `nBits` određuje ciljnu težinu za pronalaženje *hash* bloka, niža vrijednost predstavlja veću poteškoću jer se traži *hash* ispod mrežnog cilja, a veća vrijednost predstavlja manju težinu. Formula 3.1. prikazuje način na koji parametar `nBits` određuje `powLimit` korištenjem vrijednosti 0x1d00ffff kao primjer.

$$0x1d00ffff = 256^{(0x1d-3)} \cdot 0x00ffff = 256^{(29-3)} \cdot 65535 = 256^{26} \cdot (256^2 - 1)$$
$$= 0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff \quad (3\text{-}1)$$

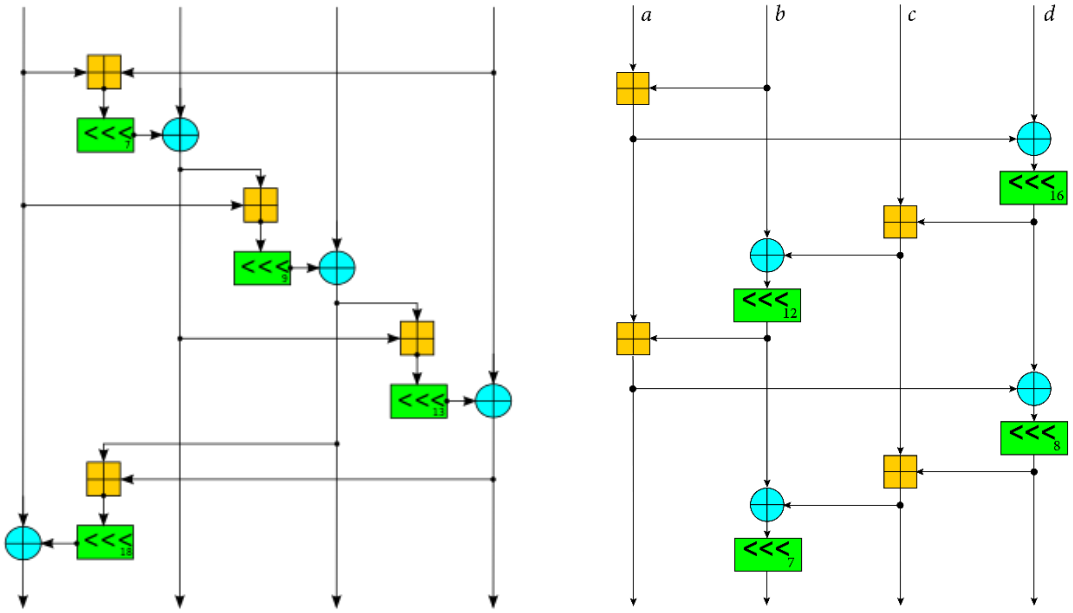
Težina pronalaska novog *hash*-a je proporcionalna s vremenom potrebnim za njegov pronalazak. Promjenom težine s 0x1d00ffff na 0x1e00ffff vrijeme potrebno za generiranje *genesis* bloka se znatno smanjilo. Nakon promjene *genesis* bloka i još nekih osnovnih stvari (`pchMessageStart`, mijenjanje porta, `base58` ključeva i ekstenzija, `blocktime, timestamp, PoWlimit, seed`, itd.) ponovno je pokrenut `./feritcoind`. Zbog promjene *genesis* bloka, zahtijeva se ponovno ndeksiranje kako bi se pokrenuo novi *blockchain*, tako da je potrebno pokrenuti naredbu `./feritcoind -reindex`, te kao provjeru `./feritcoin-cli getblockchaininfo`.

Da bi se rudarenje pokrenulo kreirana je datoteka `mining.sh`. Rudarenje se pokreće pomoću naredbe `./feritcoin-cli generate 1` koja se poziva velik broj puta pomoću *for* petlje. Rudarenjem se puni digitalni novčanik, jer se za svaki riješeni *hash* dobije nagrada u određenom iznosu. Pošto je *blockchain* na početku, taj iznos je 50 FRT novčića, a povećanjem *blockchain*-a nagrada se smanjuje, kao i kod Bitcoin-a. Stvaranjem stanja „na računu“ moguće je izvršavati transakcije, npr. može se na drugom računalu (ili drugoj virtualnoj mašini) pokrenuti isti kod i rudarenje, računala se mogu međusobno povezati te slati jedan drugome novce pomoću naredbe `./feritcoin-cli sendtoadress <address> <amount>` pri čemu drugo računalo može saznati svoju adresu pomoću naredbe `./feritcoin-cli getaddressinfo`. Kako bi se novac uopće mogao slati mora postojati konekcija između računala što se provjerava naredbom `./feritcoin-cli getnetworkinfo` koja pored ostalih podataka ispisuje broj konekcija koji bi u ovom slučaju trebao biti 1.

Nakon uspješnog rudarenja, potrebno je promijeniti i *hashing* algoritam čime se mijenja i *genesis* blok. Za *hashing* algoritam je odabran NeoScrypt. NeoScrpt se pojavljuje u lipnju 2014.godine, to

je PoW *hashing* algoritam temeljen na originalnom Scrypt algoritmu, ali je kriptografski jači, troši manje memorije i ASIC (engl. *Application-specific integrated circuit*) rezistentan[8] je. Nadogradnja je za cilj imala povećanje sigurnosti i bolje performanse na računalnima opće namjene, što je i glavni razlog za odabir ovog algoritma, uz održavanje usporedivih troškova i zahtijeva. Originalno je dizajniran za Feathercoin. Osnova NeoScrypta je korištenje 20 rundi nereducirane Salse20 i 20 rundi nereducirane ChaCha20 [20].

Salsa20 i blisko povezana ChaCha20 su tokovne šifre (engl. *cipher stream*) koje je razvio Daniel J. Bernstein. Salsa20 razvijena je 2005.godine, a ChaCha20, koja je modifikacija Salse20, objavljena je 2008. Obje šifre se temelje na pseudo-slučajnoj funkciji koja se temelji na operacijama dodavanje-rotiranje-XOR (engl. *add-rotate-XOR*) ili skraćeno ARX. Osnovna funkcija preslikava 128-bitni ključ, 64-bitni *nonce* i 64-bitni brojač u 256-bitni blok toka ključeva. Krajnji rezultat NeoScrypt algoritma dobije se dovođenjem izlaza iz Salsa20 i ChaCha20 funkcija na ulaz XOR funkcije [21]. Na slici 3.1. su prikazane četvrt-kružne funkcije (engl. *quarter-round functions*) pojedinih šifri toka koje se spajaju paralelno po četiri puta čime se dobije čitava runda. $\boxplus$ predstavlja zbrajanje, ‹‹‹ rotaciju te $\oplus$ predstavlja XOR funkciju [22].



Slika 1.3. Salsa20 četvrt-kružna funkcija i ChaCha20 četvrt-kružna funkcija [21] [22]

Može se zaključiti kako je NeoScrypt složeniji od SHA-256 algoritma, tj. procesno je zahtjevniji, sporiji i memorijski intenzivniji, što kao rezultat ima povećanu sigurnost. Trend korištenja

[8] Svojstvo dizajna u kripto valutama i kriptografskim algoritmima čiji je cilj otežati ili učiniti nepraktičnim razvoj specijaliziranog hardvera (ASIC) za rudarenje ili izvođenje specifičnih računalnih zadataka.

NeoScrypt-a kao *hashing* algoritma je u porastu, trenutno je na njemu temeljeno 47 kripto valuta, dok je na SHA-256 temeljeno njih 593 [23].

Izvorni kod NeoScrypt-a je dostupan na GitHub-u [24]. Datoteke `neoscrypt.h` i `neoscrypt.c` su dodane u `FERITcoin`, te ih je potrebno dodati i u `Makefile.am` u `FERITcoin/src/`. *Hashing* algoritam se mijenja u funkciji `GetHash()` u `block.cpp` datoteci koja se nalazi u `FERITcoin/src/primitives/`. `neoscrypt.c` je preveden u C++ što je stvaralo neke probleme i zbog čega se kod morao izgraditi iz nule. Pored promjene *hashing* algoritma, dodan je i *premining* koji korisniku, ukoliko postoji manje od 10 novčića u *blockchain*-u, dodjeljuje 1000FRT što je definirano u datoteci `validation.cpp` u `FERITcoin/src/`.

Zbog promjene *hashing* algoritma u NeoScrypt se za generiranje *genesis* bloka više ne može koristiti GenesisH0 alat koji je baziran na SHA-256, zbog čega je korišten sljedeći kod u `chainparams.cpp`:

Linija	Kod
124	`genesis = CreateGenesisBlock(1687613705, 127241, 0x1e00ffff, 1, 50 * COIN);`
125	`consensus.hashGenesisBlock = genesis.GetHash();`
126	`for(; UintToArith256(genesis.GetHash()) > UintToArith256(consensus.powLimit); genesis.nNonce++){`
127	`if (genesis.nNonce % 1000 == 0){`
128	`printf("Nonce: %d\n", genesis.nNonce);`
129	`printf("Hash: %s\n", genesis.GetHash().ToString().c_str());`
130	`}`
131	`}`
132	`printf("Genesis block: %s\n Merkle root: %s\n Nonce: %d\n", genesis.GetHash().ToString().c_str(),`
133	`genesis.hashMerkleRoot.ToString().c_str(),`
134	`genesis.nNonce);`

Generirani *hash* i korijen Merkle stabla se zamjenjuju u kodu te se pokreće *blockchain*: `./feritcoind -reindex`. Ponovno se radi rudarenje kako bi se stvorilo stanje na računu i omogućile daljnje operacije. Promjenom *hashing* algoritma se promijenilo i vrijeme rudarenja koje je znatno povećano s time da težina nije mijenjana. Provjera stanja se može raditi pomoću naredbe `./feritcoin-cli getwalletinfo` kojom se dobije sljedeći ispis:

```
{
  „walletname“: „wallet.dat“,
  „walletversion“: 159900,
```

```
  „balance“: 13200.00000000,
  „unconfirmed_balance“: 0.00000000,
  „immature_balance“: 1750.00000000,
  „txcount“: 300,
  „keypoololdest“: 1687613705,
  „keypoolsize“: 999,
  „keypoolsize_hd_internal“: 1000,
  „paytxfee“: 0.00000000,
  „hdmasterkeyid“: „11028c4c390aa8b0c6bd125ee715177ad109e873“
}
```

4. TESTIRANJE

Prilikom testiranja vlastite kripto valute rađena je usporedba s Bitcoinom, odnosno usporedba performansi Bitcoin-ovog SHA-256 *hashing* algoritma i FERITcoin-ovog NeoScrypt-a. Mjerena su vremena blokova te je promatrano zauzeće memorije na dva uređaja – laptop i stolno računalo. U tablici 4.1. prikazane su osnovne hardverske i softverske značajke tih dvaju uređaja. Stolno računalo je dosta starije i slabije održavano (redovno čišćenje i hardverskog i softverskog dijela), a ima i manje slobodnog memorijskog prostora.

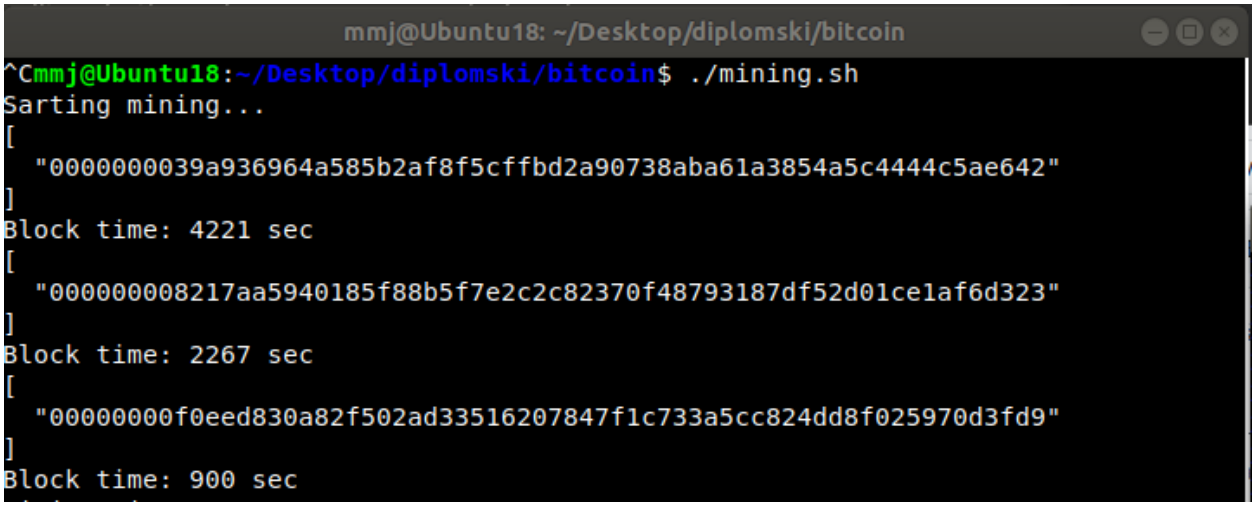
Tablica 4.1. Hardverske i softverske značajke laptopa i stolnog računala

	Laptop	Stolno računalo
Operacijski sistem (OS)	Microsoft Windows 10 Pro	Microsoft Windows 10 Pro N
Verzija OS-a	10.0.19045 N/A Build 19045	10.0.19045 N/A Build 19045
Vrsta sistema	x64	x64
Procesor	Intel® Core™ i7-4600M CPU @ 2.90GHz 2.90 GHz	Intel® Core™ i7-4771 CPU @ 3.50GHz 3.50 GHz
Slobodni RAM	3.72 GB	1.36 GB
BIOS verzija	Hewlett-Packard L70 Ver. 01.44, 08/08/2018	American Megatrends Inc. 1602, 29.10.2013.

Pored promjene *hashing* algoritma, mijenjana je i težina generiranja novih *hash*-ova, zbog čega se radi uspoređivanje NeoScrypta sa SHA-256 algoritmom i sa težinom koju koristi Bitcoin, i sa težinom koju koristi FERITcoin. Težina se određuje promjenom vrijednosti varijable `nBits` koja se zadaje kao treći parametar u funkciji `CreateGenesisBlock()` u glavnoj mreži (engl. *mainnet*). Kod Bitcoin-a je ta vrijednost postavljena na 0x1d00ffff kao što se može vidjeti u prilogu P.4.1., a smanjena je na 0x1e00ffff. Prilikom pokretanja Bitcoin-a, on nastoji već postojeći lanac skinuti s interneta kako se ne bi morao ponovno rudariti iz početka. S obzirom da računala opće namjene nemaju dovoljno memorije za pohraniti čitav Bitcoin-ov *blockchain*, ta opcija se morala onemogućiti zbog čega su sve `height` varijable u *mainnet*-u postavljene na 0, `nMinimumChainWork` je postavljen na 0, te su pobrisani svi `checkpointData` zapisi, kao i `nPruneAfterHeight`, `m_assumed_blockchain_size` i `m_assumed_chain_sate_size` varijable.

U prilogu P.4.4. je dostupna preuređena `mining.sh` skripta koja sada pored pokretanja rudarenja mjeri i vrijeme potrebno za rudarenje jednog bloka, tj. vrijeme bloka.

Nakon što su napravljene sve potrebne promjene, mjerenje je pokrenuto na 2 računala gdje su vremena blokova puno veća od prosječnih 10 minuta koje se navode na internetu, koje postižu rudari s odgovarajućom opremom. Na laptopu je srednja vrijednost vremena potrebnog za rudarenje jednog bloka skoro 33 minute, a na stolnom računalu preko 75 minuta. Za računanje srednje vrijednosti je generirano 15 blokova. Na slici 4.1. je pokazan dio ispisa koji daje skripta `mining.sh` prilikom mjerenja vremena blokova.



Slika 4.1. Primjer ispisa skripte mining.sh za originalni Bitcoin

Pokretanjem naredbe `./bitcoin-cli getblockchaininfo` moguće je dobiti informacije o trenutačnom stanju *blockchain*-a, među kojima i informaciju o zauzeću memorije. Vrijednost `size_on_disk` potrebno je podijeliti s `blocks+1` s obzirom da je *genesis* blok nulti blok. Na slici 4.2. prikazano je zauzeće memorije za 16 blokova za korištenjem SHA-256 algoritma s težinom 0x1d00ffff. Rudareni blokovi zauzimaju jednako memorije i na laptopu i na stolnome računalu, u prosjeku oko 275 kB, što odgovara očekivanim vrijednostima.

Slika 4.2. Primjer ispisa *blockchain* informacija originalnog Bitcoin-a

Smanjenem težine na 0x1e00ffff se zahtjevi za memorijom ne mijenjanju pošto oni ovise o *hashing* algoritmu koji nije mijenjan (SHA-256), ali je zbog smanjena težine vrijeme potrebno za

generiranje novog bloka drastično manje. Potrebno je bilo generirati novi *genesis* blok te promijeniti vrijednost `powLimit`-a kako bi validacija bila uspješna. Promjene su prikazane u prilogu P.4.2. Nakon pokretanja *blockchain*-a naredbom `./bitcoind -deprecatedrpc =generate`, pokrenuta je `mining.sh` skripta čiji su rezultati vidljivi na slici 4.3. Na laptopu je u prosijeku bilo potrebno oko 13 sekundi po bloku, dok je stolno računalo novi blok generiralo svake 23 sekunde.

```
mmj@Ubuntu18:~/Desktop/diplomski/bitcoin$ ./mining.sh
Sarting mining...
[
  "000000f18a664171542f212e82c35fb3f87b82363efe21ab54429a677625f99a"
]
Block time: 12 sec
[
  "000000493fc3dbe6a796b8317eaf2740c150eda06081f5d7eeb3e956b3935f59"
]
Block time: 34 sec
[
  "00000069474ed6f8f1894cccc5ad5dc69afcef8055c4be044f3cfe13098af55c"
]
Block time: 3 sec
[
  "000000135b74a85f2bc23cdea67c5e5b875e250a1d07299cf1f606d004e81157"
]
Block time: 14 sec
```

Slika 4.3. Primjer ispisa skripte `mining.sh` za SHA-256 smanjene težine

Prilikom kreiranja FERITcoin-a mijenjan je i *hashing* u NeoScrypt koji je memorijski intenzivniji, složeniji i kriptografski sigurniji od SHA-256. Iako težina ostaje 0x1e00ffff, promjenom *hashing* algoritma mijenja se *genesis* blok. Zbog promjene `pszTimestamp`-a i ostalih varijabli kao što je prikazano u prilogu P.4.3., pored *mainnet*-a, potrebno je generirati i nove blokove za *testnet* i *regtestnet*. Kako bi se pri kreiranju blokova koristio NeoScrypt, potrebno je funkciju `GetHash()` iz datoteke `src/primitives/block.cpp` preurediti:

Linija Kod

```
14      uint256 CBlockHeader::GetHash() const
15      {
16         uint26 hash;
17         unsigned int profile = 0x0;
18         neoscrypt((unsigned char *)&nVersion, (unsigned char *)&hash, profile);
19         return hash;
20      }
```

U usporedbi s Bitcoin-om jednake težine generiranja novih blokova, FERITcoin temeljen na NeoScryptu je dosta sporiji. Na laptopu je za generiranje bloka potrebno 2.5 minute, dok je na stolnom računalu potrebno nešto ispod 6 minuta, što je u usporedbi s 13 i 23 sekunde potrebne za

generiranje bloka s SHA-256 dosta velika razlika. NeoScrypt je dizajniran s namjerom da bude sporiji i memorijski intenzivniji što je poželjno za povećanje sigurnosti. Promatrajući funkcije runda koje koriste SHA-256 i NeoScrypt te broj rundi kroz koje podatci prolaze, velike razlike u vremenima blokova su bile očekivane. Na slici 4.4. je prikazan dio mjerenja vremena blokova obavljen za FERITcoin.

```
mmj@Ubuntu18:~/Desktop/diplomski/FERITcoin$ ./mining.sh
Starting mining...
[
"0x00000a70e195718eb947c397ac357a822ae254e4c68e11eb656cded336aec27e"
]
Block time: 203 sec
[
"0x00000d2054abff9c5bf3f13270563c85db152a777f72352128a4ce5ee8e3d53a"
]
Block time: 114 sec
[
"0x00000e0cadee2fa74f2fd89749e1fdfc17d377c1aff3143ae7b5e19b8fe65035"
]
Block time: 93 sec
[
"0x000005be6ae000d98d9c53aea8c5e3769a1552a18d317455a98271edef3f668a"
]
Block time: 174 sec
```

Slika 4.4. Primjer ispisa skripte mining.sh za FERITcoin

Jedan blok generiran korištenjem NeoScrypt-a zauzima prosječno 434kB memorije, što je gotovo pa duplo u odnosu na blok generiran SHA-256 algoritmom. Pored promjene samog `pszTimestamp`-a koji je veći što utječe na duljinu izlaza prilikom generiranja *hash*-a, NeoScrypt, za razliku od SHA-256, nema izlaz fiksne duljine. Duljina izlaza se povećava značajkama kao što su soljenje (engl. *salting*) i protezanje ključa (engl. *key stretching*). *Salting* je proces dodavanja slučajnih vrijednosti na ulaz prije *hash*-iranja i procesiranja. *Key stretching* je postupak opetovane obrade ulaznih podataka kako bi se usporio proces *hash*-iranja. Na slici 4.5. prikazano je zauzeće memorije korištenjem NeoScrypta.

```
mmj@Ubuntu18:~/Desktop/diplomski/FERITcoin/src$ ./feritcoin-cli getblockchaininfo
{
  "chain": "main",
  "blocks": 15,
  "headers": 15,
  "bestblockhash": "000009a70ba2b8a28084029a6fc36ebd51155c6e114e211d13a57cb72d81ac42",
  "difficulty": 0.00390625,
  "mediantime": 1720003211,
  "verificationprogress": 1.915426854245213e-08,
  "initialblockdownload": false,
  "chainwork": "0000000000000000000000000000000000000000000000000000000011001100",
  "size_on_disk": 6943,
```

Slika 4.5. Primjer ispisa *blockchain* informacija FERITcoin-a

U tablici 4.1. su prikazani rezultati mjerenja pomoću kojih se računala srednja vrijednost vremena bloka te su izračunate apsolutne i relativne razlike između laptopa i stolnog računala.

Tablica 4.1. Rezultati mjerenja vremena blokova

Blocktime - Originalni Bitcoin [s]							Srednja vrijednost	Apsolutna i relativna razlika
Laptop	Blok	1	2	3	4	5	1968.467 (32.81min)	A: 2564.400 (42.74min) R: 130.274%
	Vrijeme	4221	2267	900	136	2415		
	Blok	6	7	8	9	10		
	Vrijeme	1871	2394	490	253	1490		
	Blok	11	12	13	14	15		
	Vrijeme	4799	1025	288	4789	2189		
Stolno računalo	Blok	1	2	3	4	5	4532.867 (75.55min)	
	Vrijeme	2245	618	7387	242	596		
	Blok	6	7	8	9	10		
	Vrijeme	2145	1076	5042	5356	5042		
	Blok	11	12	13	14	15		
	Vrijeme	5356	20002	6787	5023	1076		
Blocktime - Bitcoin smanjene težine [s]							Srednja vrijednost	Apsolutna i relativna razlika
Laptop	Blok	1	2	3	4	5	12.933 (0.215min)	A: 9.734 (0.162min) R: 75.265%
	Vrijeme	12	34	3	14	2		
	Blok	6	7	8	9	10		
	Vrijeme	8	7	41	19	16		
	Blok	11	12	13	14	15		
	Vrijeme	1	27	2	6	2		
Stolno računalo	Blok	1	2	3	4	5	22.667 (0.377min)	
	Vrijeme	31	46	13	1	46		
	Blok	6	7	8	9	10		
	Vrijeme	24	4	5	15	9		
	Blok	11	12	13	14	15		
	Vrijeme	35	16	39	8	48		
Blocktime – FERITcoin [s]							Srednja vrijednost	Apsolutna i relativna razlika
Laptop	Blok	1	2	3	4	5	155.067 (2.584min)	A: 190.732 (3.179min) R: 123.000%
	Vrijeme	203	114	93	174	196		
	Blok	6	7	8	9	10		
	Vrijeme	227	138	145	82	159		
	Blok	11	12	13	14	15		
	Vrijeme	134	98	227	164	172		
Stolno računalo	Blok	1	2	3	4	5	345.8 (5.763min)	
	Vrijeme	273	315	295	372	412		
	Blok	6	7	8	9	10		
	Vrijeme	272	394	356	422	314		
	Blok	11	12	13	14	15		
	Vrijeme	327	312	437	299	387		

Iako bi NeoScrypt trebao imati bolje performanse na računalnim hardverima opće namjene od Scrypt-a, pa samim time i od SHA-256, mjerenja su pokazala da zbog svoje složenosti traži veće zahtjeve za memorijom, a i vremenom izvođenja u odnosu na SHA-256. Zbog te složenosti je bilo potrebno smanjiti težinu generiranja novih blokova, a samim time i PoW limita kako bi se *blockchain* mogao graditi u nekom razumom vremenu.

5. ZAKLJUČAK

Popularnost kripto valuta iz dana u dan raste, pogotovo pojavom „kripto milijunaša“ zbog čega većina na njih gleda kao na sredstvo zarade, ali često nisu svjesni što je sve potrebno za održavanje kripto valuta „na životu“ i zašto postoji toliki promet kripto valutama. Iako je danas plaćanje kripto valutama u trendu te je omogućeno plaćanje, primjera radi, u kafićima, restoranima i slično, kod većine transakcija je legalnost upitna. Pored legalnosti, veliki rizik za ulaganje u kripto valute čini njihova nestabilnost, jer nemaju osnovu na kojoj grade svoju vrijednost, nego „vrijede samo onoliko koliko je netko spreman platiti“.

Pojava ekološki prihvatljivih kripto valuta je pozitivna stvar, ali teško da će njihova popularnost dostići onu od Bitcoin-a, zbog čega će utjecaj na okoliš biti tema još dugi niz godina. Iako se kripto valute smatraju neovisnim o državi i bankama, uvođenje sankcija u pojedinim državama dokazala je suprotno. Većina država se bavi legalnošću, odnosno zaobilaženjem plaćanja poreza i pranjem novca, ali i utjecaj na okoliš je tema i razmatranje ekološki prihvatljivih valuta. Zabrana kripto valuta temeljenih na PoW konsenzusu je vjerojatno jedini način da se Bitcoin skine s trona, ukoliko i on ne pređe na PoS koji ima manju emisiju ugljičnog dioksida.

Trenutno postoji skoro 23.000 *altcoin*-ova, od kojih je većina kreirana na osnovu Bitcoin-a. Sam proces razvoja vlastite kripto valute – FERITcoin-a, iako se strukturno ne mijenja puno, je jako zahtijevan, pogotovo vremenski, jer generiranje *genesis* bloka, rudarenje, itd. može trajati i po par sati, pa čak i po par dana. Zamjenom SHA-256 algoritma s NeoScryptom se povećalo vrijeme potrebno za generiranje novog bloka i memorija koju taj blok zauzima. Razlog tomu je veća složenost NeoScrypta koja se temelji na povećanoj memorijskoj intenzivnošću i namjernom usporavanju procesa *hash*-iranja. Pored razlike u vremenima blokova zbog promjene *hashing* algoritma i težine, primijećena je i razlika prilikom rudarenja na različitim uređajima gdje su istaknute njihove hardverske značajke. Prilikom razvoja je stečeno dublje razumijevanje osnovnih tehnologija koje se koriste u kripto valutama, povezana je teorija s konkretnim slučajevima i primjerima u samome kodu.

LITERATURA

[1] M. Milutinović, http://scindeks.ceon.rs/Article.aspx?artid=0350-137X1801105M, SCIndex, lipanj 2023.

[2] Jamie Redman, https://news.bitcoin.com/altcoins-why-over-5000/, Bitcoin.com, lipanj 2023.

[3] Bitinfocharts, https://bitinfocharts.com/, lipanj 2023.

[4] Narayanan, Arvind; Bonneau, Joseph; Felten, Edward; Miller, Andrew; Goldfeder, Steven (2016). *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton, New Jersey: Princeton University Press

[5] Wikipedia, https://en.wikipedia.org/wiki/Blockchain, lipanj 2023.

[6] Wikipedia, https://en.wikipedia.org/wiki/Byzantine_fault, lipanj 2023.

[7] J. Frankenfield, https://www.investopedia.com/terms/p/proof-work.asp, Investopedia, lipanj 2023.

[8] J. Frankenfield, https://www.investopedia.com/terms/p/proof-stake-pos.asp, Investopedia, lipanj 2023.

[9] Cointelegraph, https://cointelegraph.com/learn/ethereum-sharding-a-beginners-guide-to-blockchain-sharding, kolovoz 2023.

[10] Slickcharts, https://www.slickcharts.com/currency, lipanj 2023.

[11] YCharts, https://ycharts.com/indicators/bitcoin_blockchain_size, lipanj 2023.

[12] Golden , https://golden.com/wiki/SHA-256-XKEJ8AB, kolovoz 2023.

[13] Zeyad A Al-Odat, https://www.researchgate.net/figure/General-architecture-to-compute-the-SHA-256-hash-function_fig1_335095939, Researchgate, rujan 2023.

[14] Golden, https://golden.com/wiki/SHA-256-XKEJ8AB, kolovoz 2023.

[15] Somitra Kumar Sanadhya, https://www.researchgate.net/figure/Round-function-of-SHA-256-hash-function_fig1_220335111, Researchgate, kolovoz 2023.

[16] T. Kim, https://www.cnbc.com/2017/07/26/billionaire-investor-marks-who-called-the-dotcom-bubble-says-bitcoin-is-a-pyramid-scheme.html, CNBC, lipanj 2023.

[17] Bitcoin, https://github.com/bitcoin/bitcoin/blob/master/doc/build-unix.md, lipanj 2023.

[18] Martina-Magdalena Jukić, https://github.com/jukmmartina/FERITcoin, GitHub, lipanj 2023.

[19] Lauri Hartikka, https://github.com/lhartikk/GenesisH0, GitHub, lipanj 2023.

[20] John Doering, http://phoenixcoin.org/archive/neoscrypt_v1.pdf, kolovoz 2023.

[21] Wikipedia https://en.wikipedia.org/wiki/Salsa20, kolovoz 2023.

[22] DOCS.RS, https://docs.rs/chacha20/latest/chacha20/, kolovoz 2023.

[23] Cryptoval, https://cryptorival.com/algorithms/, kolovoz 2023.

[24] John Doering, https://github.com/ghostlander/NeoScrypt, GitHub, lipanj 2023.

[25] Elliott Minns, https://educationecosystem.com/elliottminns/l3Xa0-how-to-create-your-own-cryptocurrency-in-c/, Education Ecosystem, lipanj 2023.

SAŽETAK

Ključne riječi: bitcoin, *blockchain*, *genesis* blok, kripto valuta, metode konsensusa

U ovom diplomskom radu opisana je povijest i razvoj kripto valuta te tehnologija koje se koriste pri razvoju (*blockchain*, metode konsenzusa, digitalni novčanik). Tehnologije su konkretno opisane na primjeru Bitcoin-a na osnovu kojega je i razvijena vlastita kripto valuta u programskom jeziku C++. Za razvoj vlastite kripto valute bilo je potrebno generirati novi *genesis* blok (prvi blok u *blockhain*-u) pomoću GenesisH0 alata te je promijenjena i težina generiranja novog bloka. Nakon uspješnog generiranja *genesis* bloka i pokretanja *blockchain*-a, promijenjen je *hashing* algoritam iz SHA-256 u NeoScrypt. Pokrenuto je rudarenje gdje je potvrđeno kako je za rudarenje bloka potrebno oko dvije i pol minute i kako jedan blok zauzima oko 500 kB memorije. Uspoređene su performanse SHA-256 i NeoScrypt algoritma čime su teorijske podloge povezane s konkretnim primjerima.

CRYPTOCURRENCY IMPLEMENTATION IN C++ PROGRAMMING LANGUAGE

Key words: bitcoin, blockchain, genesis block, cryptocurrency, consensus method

Abstract: This thesis describes the history and development of cryptocurrencies and the technologies used in their development (blockchain, consensus methods, wallet). The technologies are described on the example of Bitcoin, based on Bitcoin the own crypto currency was developed in the programming language C++. To develop your own cryptocurrency, it was necessary to generate a new genesis block (the first block in the blockchain) using the GenesisH0 tool, the difficulty of generating a new block wes also changed. After successfully generating the genesis block and starting the blockchain, the hashing algorithm was changed from SHA-256 to NeoScrypt. Mining was started, where it was confirmed that mining a block takes about two and a half minutes and that one block occupies around 500 kB of memory. Performances of the SHA-256 and NeoScrypt algorithms are compared, which connects theoretical foundations with concrete examples.

ŽIVOTOPIS

Martina-Magdalena Jukić, studentica 2.godine diplomskog studija Komunikacije i informacijske tehnologije na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija, rođena je 02.07.1999.godine u Žepču, FBiH. U Žepču je pohađala osnovnu i srednju školu, a svoje obrazovanje nastavila je u Osijeku. Tijekom školovanja dobitnica je brojnih nagrada, kao i dvostruki dobitnik Nagrade za postignuti uspjeh u studiranju. Od ožujka 2021.godine do ožujka 2023. godine radila je u osječkoj firmi Orqa kao embedded developer, a od ožujka 2023. radi kao backend developer u Glooko-u.

_____

Potpis autora

PRILOZI

Prilog P.3.1. Primjer ispisa komande `bitcoin-cli geblockchaininfo`


```
{
  "chain": "main",
  "blocks": 0,
  "headers": 0,
  "bestblockhash":
"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
  "difficulty": 1,
  "mediantime": 1231006505,
  "verificationprogress": 2.952906182569511e-09
  "initialblockdownload": true,
  "chainwork":
"0000000000000000000000000000000000000000000000000100010001",
  "size_on_disk": 293,
  "pruned": false,
  "softworks": [
    {
      "id": "bip34",
      "version": 2,
      "reject": {
        "status": true
      }
    },
    {
      "id": "bip66",
      "version": 3,
      "reject": {
        "status": true
      }
    },
    {
      "id": "bip65",
      "version": 4,
      "reject": {
```

```
        "status": true
      }
    }
  ],
  "bip9_softworks": {
    "csv": {
      "status": "defined",
      "startTime": 1462060800,
      "timeout": 1493596800,
      "since": 0
    },
    "segwit": {
      "status": "defined",
      "startTime": 1479168000
      "timeout": 1510704000,
      "since": 0
    }
  },
  "warnings": ""
}
```

Prilog P.4.1. Datoteka `src/chainparams.cpp` za rudarenje Bitcoin-a s originalnom težinom generiranja *hash*-ova

```
// Copyright (c) 2010 Satoshi Nakamoto
// Copyright (c) 2009-2018 The Bitcoin Core developers
// Distributed under the MIT software license, see the accompanying
// file COPYING or http://www.opensource.org/licenses/mit-license.php.

#include <chainparams.h>

#include <chainparamsseeds.h>
#include <consensus/merkle.h>
#include <tinyformat.h>
#include <util/system.h>
#include <util/strencodings.h>
#include <versionbitsinfo.h>

#include <assert.h>

#include <boost/algorithm/string/classification.hpp>
#include <boost/algorithm/string/split.hpp>

#include <arith_uint256.h>

static CBlock CreateGenesisBlock(const char* pszTimestamp, const CScript&
genesisOutputScript, uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t
nVersion, const CAmount& genesisReward)
{
    CMutableTransaction txNew;
    txNew.nVersion = 1;
    txNew.vin.resize(1);
    txNew.vout.resize(1);
    txNew.vin[0].scriptSig = CScript() << 486604799 << CScriptNum(4) <<
std::vector<unsigned char>((const unsigned char*)pszTimestamp, (const
unsigned char*)pszTimestamp + strlen(pszTimestamp));
    txNew.vout[0].nValue = genesisReward;
    txNew.vout[0].scriptPubKey = genesisOutputScript;

    CBlock genesis;
    genesis.nTime    = nTime;
    genesis.nBits    = nBits;
    genesis.nNonce   = nNonce;
    genesis.nVersion = nVersion;
    genesis.vtx.push_back(MakeTransactionRef(std::move(txNew)));
    genesis.hashPrevBlock.SetNull();
    genesis.hashMerkleRoot = BlockMerkleRoot(genesis);
    return genesis;
}

/**
 * Build the genesis block. Note that the output of its generation
 * transaction cannot be spent since it did not originally exist in the
 * database.
 *
 * CBlock(hash=000000000019d6, ver=1, hashPrevBlock=00000000000000,
hashMerkleRoot=4a5e1e, nTime=1231006505, nBits=1d00ffff, nNonce=2083236893,
vtx=1)
 *   CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
 *     CTxIn(COutPoint(000000, -1), coinbase
04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6
```

```
f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73
)
 *     CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
 *   vMerkleTree: 4a5e1e
 */
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_t
nBits, int32_t nVersion, const CAmount& genesisReward)
{
    const char* pszTimestamp = "The Times 03/Jan/2009 Chancellor on brink of
second bailout for banks";
    const CScript genesisOutputScript = CScript() <<
ParseHex("04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb64
9f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f") <<
OP_CHECKSIG;
    return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime,
nNonce, nBits, nVersion, genesisReward);
}

/**
 * Main network
 */
class CMainParams : public CChainParams {
public:
    CMainParams() {
        strNetworkID = "main";
        consensus.nSubsidyHalvingInterval = 210000;
        consensus.BIP16Exception =
uint256S("0x00000000000002dc756eebf4f49723ed8d30cc28a5f108eb94b1ba88ac4f9c22"
);
        consensus.BIP34Height = 0;
        consensus.BIP34Hash =
uint256S("0x000000000000024b89b42a942fe0d9fea3bb44ab7bd1b19115dd6a759c0808b8"
);
        consensus.BIP65Height = 0; //
00000000000000004c2b624ed5d7756c508d90fd0da2c7c679febfa6c4735f0
        consensus.BIP66Height = 0; //
00000000000000000379eaa19dce8c9b722d46ae6a57c2f1a988119488b50931
        consensus.powLimit =
uint256S("00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 14 * 24 * 60 * 60; // two weeks
        consensus.nPowTargetSpacing = 10 * 60;
        consensus.fPowAllowMinDifficultyBlocks = false;
        consensus.fPowNoRetargeting = false;
        consensus.nRuleChangeActivationThreshold = 1916; // 95% of 2016
        consensus.nMinerConfirmationWindow = 2016; // nPowTargetTimespan /
nPowTargetSpacing
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
1199145601; // January 1, 2008
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
1230767999; // December 31, 2008

        // Deployment of BIP68, BIP112, and BIP113.
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime =
1462060800; // May 1st, 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
1493596800; // May 1st, 2017

        // Deployment of SegWit (BIP141, BIP143, and BIP147)
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
```

```
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
1479168000; // November 15th, 2016.
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
1510704000; // November 15th, 2017.

        // The best chain should have at least this much work.
        consensus.nMinimumChainWork = uint256S("0x00");

        // By default assume that the signatures in ancestors of this block
are valid.
        consensus.defaultAssumeValid =
uint256S("0x00000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee"
); //563378

        /**
         * The message start string is designed to be unlikely to occur in
normal data.
         * The characters are rarely used upper ASCII, not valid as UTF-8,
and produce
         * a large 32-bit integer with any alignment.
         */
        pchMessageStart[0] = 0xf9;
        pchMessageStart[1] = 0xbe;
        pchMessageStart[2] = 0xb4;
        pchMessageStart[3] = 0xd9;
        nDefaultPort = 8333;

        /*
                genesis = CreateGenesisBlock(1231006505, 2147397399,
0x1d00ffff, 1, 50 * COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        for(; UintToArith256(genesis.GetHash()) >
UintToArith256(consensus.powLimit); genesis.nNonce++){
            if (genesis.nNonce % 100000 == 0){
                printf("Nonce: %d\n", genesis.nNonce);
                printf("Hash: %s\n", genesis.GetHash().ToString().c_str());
            }
        }
        printf("Genesis block: %s\n Merkle root: %s\n Nonce: %d\n",
genesis.GetHash().ToString().c_str(),

genesis.hashMerkleRoot.ToString().c_str(),

genesis.nNonce);

        assert(consensus.hashGenesisBlock ==
uint256S("0x00000000a1c93f3fa48dfa8b96eccc51c95ebdbb1ef174a774a363196c98d25a"
));
        //assert(consensus.hashGenesisBlock ==
uint256S("0x00000000aa38157b4ded8a0fac316d671647c9201948b3d069fb8491dacc7fff"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0xf7cf05c3298f8efce5025ec06985f4be005fd9ad820b17737c22416058b9e428"
));
        */

        genesis = CreateGenesisBlock(1231006505, 2083236893, 0x1d00ffff, 1,
50 * COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        for(; UintToArith256(genesis.GetHash()) >
UintToArith256(consensus.powLimit); genesis.nNonce++){
```

```
            if (genesis.nNonce % 100000 == 0){
                printf("Nonce: %d\n", genesis.nNonce);
                printf("Hash: %s\n", genesis.GetHash().ToString().c_str());
            }
        }
        printf("Genesis block: %s\n Merkle root: %s\n Nonce: %d\n",
genesis.GetHash().ToString().c_str(),

genesis.hashMerkleRoot.ToString().c_str(),

genesis.nNonce);

        assert(consensus.hashGenesisBlock ==
uint256S("0x000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f"
));
        //assert(consensus.hashGenesisBlock ==
uint256S("0x00000000aa38157b4ded8a0fac316d671647c9201948b3d069fb8491dacc7fff"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
));

        // Note that of those which support the service bits prefix, most
only support a subset of
        // possible options.
        // This is fine at runtime as we'll fall back to using them as a
oneshot if they don't support the
        // service bits we want, but we should get them updated to support
all service bits wanted by any
        // release ASAP to avoid it where possible.

        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,0);
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,5);
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,128);
        base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x88, 0xB2, 0x1E};
        base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x88, 0xAD, 0xE4};

        bech32_hrp = "bc";

        vFixedSeeds = std::vector<SeedSpec6>(pnSeed6_main, pnSeed6_main +
ARRAYLEN(pnSeed6_main));

        fDefaultConsistencyChecks = false;
        fRequireStandard = true;
        fMineBlocksOnDemand = false;

        checkpointData = {
            {
            }
        };

        chainTxData = ChainTxData{
            // Data from rpc: getchaintxstats 4096
000000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee
            /* nTime    */ 1550374134,
            /* nTxCount */ 383732546,
            /* dTxRate  */ 3.685496590998308
        };

        /* disable fallback fee on mainnet */
        m_fallback_fee_enabled = false;
```

```
    }
};

/**
 * Testnet (v3)
 */
class CTestNetParams : public CChainParams {
public:
    CTestNetParams() {
        strNetworkID = "test";
        consensus.nSubsidyHalvingInterval = 210000;
        consensus.BIP16Exception =
uint256S("0x00000000dd30457c001f4095d208cc1296b0eed002427aa599874af7a432b105"
);
        consensus.BIP34Height = 21111;
        consensus.BIP34Hash =
uint256S("0x0000000023b3a96d3484e5abb3755c413e7d41500f8e2a5c3f0dd01299cd8ef8"
);
        consensus.BIP65Height = 581885; //
00000000007f6655f22f98e72ed80d8b06dc761d5da09df0fa1dc4be4f861eb6
        consensus.BIP66Height = 330776; //
000000002104c8c45e99a8853285a3b592602a3ccde2b832481da85e9e4ba182
        consensus.powLimit =
uint256S("00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 14 * 24 * 60 * 60; // two weeks
        consensus.nPowTargetSpacing = 10 * 60;
        consensus.fPowAllowMinDifficultyBlocks = true;
        consensus.fPowNoRetargeting = false;
        consensus.nRuleChangeActivationThreshold = 1512; // 75% for
testchains
        consensus.nMinerConfirmationWindow = 2016; // nPowTargetTimespan /
nPowTargetSpacing
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
1199145601; // January 1, 2008
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
1230767999; // December 31, 2008

        // Deployment of BIP68, BIP112, and BIP113.
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime =
1456790400; // March 1st, 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
1493596800; // May 1st, 2017

        // Deployment of SegWit (BIP141, BIP143, and BIP147)
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
1462060800; // May 1st 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
1493596800; // May 1st 2017

        // The best chain should have at least this much work.
        consensus.nMinimumChainWork =
uint256S("0x00000000000000000000000000000000000000000007dbe94253893cbd463"
);

        // By default assume that the signatures in ancestors of this block
are valid.
```

```
        consensus.defaultAssumeValid =
uint256S("0x0000000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7cfc2b4c75"
); //1354312

        pchMessageStart[0] = 0x0b;
        pchMessageStart[1] = 0x11;
        pchMessageStart[2] = 0x09;
        pchMessageStart[3] = 0x07;
        nDefaultPort = 18333;
        nPruneAfterHeight = 1000;
        m_assumed_blockchain_size = 30;
        m_assumed_chain_state_size = 2;

        genesis = CreateGenesisBlock(1296688602, 414098458, 0x1d00ffff, 1, 50
* COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        assert(consensus.hashGenesisBlock ==
uint256S("0x000000000933ea01ad0ee984209779baaec3ced90fa3f408719526f8d77f4943"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
));

        vFixedSeeds.clear();
        vSeeds.clear();
        // nodes with support for servicebits filtering should be at the top
        vSeeds.emplace_back("testnet-seed.bitcoin.jonasschnelli.ch");
        vSeeds.emplace_back("seed.tbtc.petertodd.org");
        vSeeds.emplace_back("seed.testnet.bitcoin.sprovoost.nl");
        vSeeds.emplace_back("testnet-seed.bluematt.me"); // Just a static
list of stable node(s), only supports x9

        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,111);
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,196);
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,239);
        base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x35, 0x87, 0xCF};
        base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x35, 0x83, 0x94};

        bech32_hrp = "tb";

        vFixedSeeds = std::vector<SeedSpec6>(pnSeed6_test, pnSeed6_test +
ARRAYLEN(pnSeed6_test));

        fDefaultConsistencyChecks = false;
        fRequireStandard = false;
        fMineBlocksOnDemand = false;


        checkpointData = {
            {
                {546,
uint256S("000000002a936ca763904c3c35fce2f3556c559c0214345d31b1bcebf76acb70")}
,
            }
        };

        chainTxData = ChainTxData{
            // Data from rpc: getchaintxstats 4096
0000000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7cfc2b4c75
            /* nTime    */ 1531929919,
            /* nTxCount */ 19438708,
```

```
            /* dTxRate  */ 0.626
        };

        /* enable fallback fee on testnet */
        m_fallback_fee_enabled = true;
    }
};

/**
 * Regression test
 */
class CRegTestParams : public CChainParams {
public:
    explicit CRegTestParams(const ArgsManager& args) {
        strNetworkID = "regtest";
        consensus.nSubsidyHalvingInterval = 150;
        consensus.BIP16Exception = uint256();
        consensus.BIP34Height = 500; // BIP34 activated on regtest (Used in
functional tests)
        consensus.BIP34Hash = uint256();
        consensus.BIP65Height = 1351; // BIP65 activated on regtest (Used in
functional tests)
        consensus.BIP66Height = 1251; // BIP66 activated on regtest (Used in
functional tests)
        consensus.powLimit =
uint256S("7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 14 * 24 * 60 * 60; // two weeks
        consensus.nPowTargetSpacing = 10 * 60;
        consensus.fPowAllowMinDifficultyBlocks = true;
        consensus.fPowNoRetargeting = true;
        consensus.nRuleChangeActivationThreshold = 108; // 75% for testchains
        consensus.nMinerConfirmationWindow = 144; // Faster than normal for
regtest (144 instead of 2016)
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
0;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
Consensus::BIP9Deployment::ALWAYS_ACTIVE;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;

        // The best chain should have at least this much work.
        consensus.nMinimumChainWork = uint256S("0x00");

        // By default assume that the signatures in ancestors of this block
are valid.
        consensus.defaultAssumeValid = uint256S("0x00");

        pchMessageStart[0] = 0xfa;
        pchMessageStart[1] = 0xbf;
        pchMessageStart[2] = 0xb5;
        pchMessageStart[3] = 0xda;
        nDefaultPort = 18444;
        nPruneAfterHeight = 1000;
```

```
        m_assumed_blockchain_size = 0;
        m_assumed_chain_state_size = 0;

        UpdateVersionBitsParametersFromArgs(args);

        genesis = CreateGenesisBlock(1296688602, 2, 0x207fffff, 1, 50 *
COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        assert(consensus.hashGenesisBlock ==
uint256S("0x0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
));

        vFixedSeeds.clear(); //!< Regtest mode doesn't have any fixed seeds.
        vSeeds.clear();      //!< Regtest mode doesn't have any DNS seeds.

        fDefaultConsistencyChecks = true;
        fRequireStandard = false;
        fMineBlocksOnDemand = true;

        checkpointData = {
            {
                {0,
uint256S("0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206")}
,
            }
        };

        chainTxData = ChainTxData{
            0,
            0,
            0
        };

        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,111);
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,196);
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,239);
        base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x35, 0x87, 0xCF};
        base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x35, 0x83, 0x94};

        bech32_hrp = "bcrt";

        /* enable fallback fee on regtest */
        m_fallback_fee_enabled = true;
    }

    /**
     * Allows modifying the Version Bits regtest parameters.
     */
    void UpdateVersionBitsParameters(Consensus::DeploymentPos d, int64_t
nStartTime, int64_t nTimeout)
    {
        consensus.vDeployments[d].nStartTime = nStartTime;
        consensus.vDeployments[d].nTimeout = nTimeout;
    }
    void UpdateVersionBitsParametersFromArgs(const ArgsManager& args);
};
```

```
void CRegTestParams::UpdateVersionBitsParametersFromArgs(const ArgsManager&
args)
{
    if (!args.IsArgSet("-vbparams")) return;

    for (const std::string& strDeployment : args.GetArgs("-vbparams")) {
        std::vector<std::string> vDeploymentParams;
        boost::split(vDeploymentParams, strDeployment,
boost::is_any_of(":"));
        if (vDeploymentParams.size() != 3) {
            throw std::runtime_error("Version bits parameters malformed,
expecting deployment:start:end");
        }
        int64_t nStartTime, nTimeout;
        if (!ParseInt64(vDeploymentParams[1], &nStartTime)) {
            throw std::runtime_error(strprintf("Invalid nStartTime (%s)",
vDeploymentParams[1]));
        }
        if (!ParseInt64(vDeploymentParams[2], &nTimeout)) {
            throw std::runtime_error(strprintf("Invalid nTimeout (%s)",
vDeploymentParams[2]));
        }
        bool found = false;
        for (int j=0; j < (int)Consensus::MAX_VERSION_BITS_DEPLOYMENTS; ++j)
{
            if (vDeploymentParams[0] == VersionBitsDeploymentInfo[j].name) {
                UpdateVersionBitsParameters(Consensus::DeploymentPos(j),
nStartTime, nTimeout);
                found = true;
                LogPrintf("Setting version bits activation parameters for %s
to start=%ld, timeout=%ld\n", vDeploymentParams[0], nStartTime, nTimeout);
                break;
            }
        }
        if (!found) {
            throw std::runtime_error(strprintf("Invalid deployment (%s)",
vDeploymentParams[0]));
        }
    }
}

static std::unique_ptr<const CChainParams> globalChainParams;

const CChainParams &Params() {
    assert(globalChainParams);
    return *globalChainParams;
}

std::unique_ptr<const CChainParams> CreateChainParams(const std::string&
chain)
{
    if (chain == CBaseChainParams::MAIN)
        return std::unique_ptr<CChainParams>(new CMainParams());
    else if (chain == CBaseChainParams::TESTNET)
        return std::unique_ptr<CChainParams>(new CTestNetParams());
    else if (chain == CBaseChainParams::REGTEST)
        return std::unique_ptr<CChainParams>(new CRegTestParams(gArgs));
    throw std::runtime_error(strprintf("%s: Unknown chain %s.", __func__,
chain));
}
```

```
void SelectParams(const std::string& network)
{
    SelectBaseParams(network);
    globalChainParams = CreateChainParams(network);
}
```

Prilog P.4.2. Datoteka `src/chainparams.cpp` za rudarenje Bitcoin-a sa smanjenom težinom generiranja novih *hash*-ova

```
// Copyright (c) 2010 Satoshi Nakamoto
// Copyright (c) 2009-2018 The Bitcoin Core developers
// Distributed under the MIT software license, see the accompanying
// file COPYING or http://www.opensource.org/licenses/mit-license.php.

#include <chainparams.h>

#include <chainparamsseeds.h>
#include <consensus/merkle.h>
#include <tinyformat.h>
#include <util/system.h>
#include <util/strencodings.h>
#include <versionbitsinfo.h>

#include <assert.h>

#include <boost/algorithm/string/classification.hpp>
#include <boost/algorithm/string/split.hpp>

#include <arith_uint256.h>

static CBlock CreateGenesisBlock(const char* pszTimestamp, const CScript&
genesisOutputScript, uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t
nVersion, const CAmount& genesisReward)
{
    CMutableTransaction txNew;
    txNew.nVersion = 1;
    txNew.vin.resize(1);
    txNew.vout.resize(1);
    txNew.vin[0].scriptSig = CScript() << 486604799 << CScriptNum(4) <<
std::vector<unsigned char>((const unsigned char*)pszTimestamp, (const
unsigned char*)pszTimestamp + strlen(pszTimestamp));
    txNew.vout[0].nValue = genesisReward;
    txNew.vout[0].scriptPubKey = genesisOutputScript;

    CBlock genesis;
    genesis.nTime    = nTime;
    genesis.nBits    = nBits;
    genesis.nNonce   = nNonce;
    genesis.nVersion = nVersion;
    genesis.vtx.push_back(MakeTransactionRef(std::move(txNew)));
    genesis.hashPrevBlock.SetNull();
    genesis.hashMerkleRoot = BlockMerkleRoot(genesis);
    return genesis;
}

/**
 * Build the genesis block. Note that the output of its generation
 * transaction cannot be spent since it did not originally exist in the
 * database.
 *
 * CBlock(hash=000000000019d6, ver=1, hashPrevBlock=00000000000000,
hashMerkleRoot=4a5e1e, nTime=1231006505, nBits=1d00ffff, nNonce=2083236893,
vtx=1)
 *   CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
 *     CTxIn(COutPoint(000000, -1), coinbase
04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6
```

```
f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73
)
 *     CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
 *   vMerkleTree: 4a5e1e
 */
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_t
nBits, int32_t nVersion, const CAmount& genesisReward)
{
    const char* pszTimestamp = "The Times 03/Jan/2009 Chancellor on brink of
second bailout for banks";
    const CScript genesisOutputScript = CScript() <<
ParseHex("04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb64
9f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f") <<
OP_CHECKSIG;
    return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime,
nNonce, nBits, nVersion, genesisReward);
}

/**
 * Main network
 */
class CMainParams : public CChainParams {
public:
    CMainParams() {
        strNetworkID = "main";
        consensus.nSubsidyHalvingInterval = 210000;
        consensus.BIP16Exception =
uint256S("0x00000000000002dc756eebf4f49723ed8d30cc28a5f108eb94b1ba88ac4f9c22"
);
        consensus.BIP34Height = 0;
        consensus.BIP34Hash =
uint256S("0x000000000000024b89b42a942fe0d9fea3bb44ab7bd1b19115dd6a759c0808b8"
);
        consensus.BIP65Height = 0; //
00000000000000004c2b624ed5d7756c508d90fd0da2c7c679febfa6c4735f0
        consensus.BIP66Height = 0; //
00000000000000000379eaa19dce8c9b722d46ae6a57c2f1a988119488b50931
        consensus.powLimit =
uint256S("0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 14 * 24 * 60 * 60; // two weeks
        consensus.nPowTargetSpacing = 10 * 60;
        consensus.fPowAllowMinDifficultyBlocks = false;
        consensus.fPowNoRetargeting = false;
        consensus.nRuleChangeActivationThreshold = 1916; // 95% of 2016
        consensus.nMinerConfirmationWindow = 2016; // nPowTargetTimespan /
nPowTargetSpacing
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
1199145601; // January 1, 2008
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
1230767999; // December 31, 2008

        // Deployment of BIP68, BIP112, and BIP113.
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime =
1462060800; // May 1st, 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
1493596800; // May 1st, 2017

        // Deployment of SegWit (BIP141, BIP143, and BIP147)
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
```

```
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
1479168000; // November 15th, 2016.
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
1510704000; // November 15th, 2017.

        // The best chain should have at least this much work.
        consensus.nMinimumChainWork = uint256S("0x00");

        // By default assume that the signatures in ancestors of this block
are valid.
        consensus.defaultAssumeValid =
uint256S("0x00000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee"
); //563378

        /**
         * The message start string is designed to be unlikely to occur in
normal data.
         * The characters are rarely used upper ASCII, not valid as UTF-8,
and produce
         * a large 32-bit integer with any alignment.
         */
        pchMessageStart[0] = 0xf9;
        pchMessageStart[1] = 0xbe;
        pchMessageStart[2] = 0xb4;
        pchMessageStart[3] = 0xd9;
        nDefaultPort = 8333;


        genesis = CreateGenesisBlock(1231006505, 2896792421, 0x1e00ffff, 1,
50 * COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        for(; UintToArith256(genesis.GetHash()) >
UintToArith256(consensus.powLimit); genesis.nNonce++){
            if (genesis.nNonce % 100000 == 0){
                printf("Nonce: %d\n", genesis.nNonce);
                printf("Hash: %s\n", genesis.GetHash().ToString().c_str());
            }
        }
        printf("Genesis block: %s\n Merkle root: %s\n Nonce: %ld\n",
genesis.GetHash().ToString().c_str(),

genesis.hashMerkleRoot.ToString().c_str(),

genesis.nNonce);

        assert(consensus.hashGenesisBlock ==
uint256S("0x00000000aa38157b4ded8a0fac316d671647c9201948b3d069fb8491dacc7fff"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
));

        // Note that of those which support the service bits prefix, most
only support a subset of
        // possible options.
        // This is fine at runtime as we'll fall back to using them as a
oneshot if they don't support the
        // service bits we want, but we should get them updated to support
all service bits wanted by any
        // release ASAP to avoid it where possible.
```

```
        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,0);
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,5);
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,128);
        base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x88, 0xB2, 0x1E};
        base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x88, 0xAD, 0xE4};

        bech32_hrp = "bc";

        vFixedSeeds = std::vector<SeedSpec6>(pnSeed6_main, pnSeed6_main +
ARRAYLEN(pnSeed6_main));

        fDefaultConsistencyChecks = false;
        fRequireStandard = true;
        fMineBlocksOnDemand = false;

        checkpointData = {
            {
            }
        };

        chainTxData = ChainTxData{
            // Data from rpc: getchaintxstats 4096
000000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee
            /* nTime    */ 1550374134,
            /* nTxCount */ 383732546,
            /* dTxRate  */ 3.685496590998308
        };

        /* disable fallback fee on mainnet */
        m_fallback_fee_enabled = false;
    }
};

/**
 * Testnet (v3)
 */
class CTestNetParams : public CChainParams {
public:
    CTestNetParams() {
        strNetworkID = "test";
        consensus.nSubsidyHalvingInterval = 210000;
        consensus.BIP16Exception =
uint256S("0x00000000dd30457c001f4095d208cc1296b0eed002427aa599874af7a432b105"
);
        consensus.BIP34Height = 21111;
        consensus.BIP34Hash =
uint256S("0x0000000023b3a96d3484e5abb3755c413e7d41500f8e2a5c3f0dd01299cd8ef8"
);
        consensus.BIP65Height = 581885; //
00000000007f6655f22f98e72ed80d8b06dc761d5da09df0fa1dc4be4f861eb6
        consensus.BIP66Height = 330776; //
000000002104c8c45e99a8853285a3b592602a3ccde2b832481da85e9e4ba182
        consensus.powLimit =
uint256S("00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 14 * 24 * 60 * 60; // two weeks
        consensus.nPowTargetSpacing = 10 * 60;
        consensus.fPowAllowMinDifficultyBlocks = true;
        consensus.fPowNoRetargeting = false;
        consensus.nRuleChangeActivationThreshold = 1512; // 75% for
testchains
```

```
        consensus.nMinerConfirmationWindow = 2016; // nPowTargetTimespan /
nPowTargetSpacing
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
1199145601; // January 1, 2008
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
1230767999; // December 31, 2008

        // Deployment of BIP68, BIP112, and BIP113.
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime =
1456790400; // March 1st, 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
1493596800; // May 1st, 2017

        // Deployment of SegWit (BIP141, BIP143, and BIP147)
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
1462060800; // May 1st 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
1493596800; // May 1st 2017

        // The best chain should have at least this much work.
        consensus.nMinimumChainWork =
uint256S("0x00000000000000000000000000000000000000000000007dbe94253893cbd463"
);

        // By default assume that the signatures in ancestors of this block
are valid.
        consensus.defaultAssumeValid =
uint256S("0x0000000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7cfc2b4c75"
); //1354312

        pchMessageStart[0] = 0x0b;
        pchMessageStart[1] = 0x11;
        pchMessageStart[2] = 0x09;
        pchMessageStart[3] = 0x07;
        nDefaultPort = 18333;
        nPruneAfterHeight = 1000;
        m_assumed_blockchain_size = 30;
        m_assumed_chain_state_size = 2;

        genesis = CreateGenesisBlock(1296688602, 414098458, 0x1d00ffff, 1, 50
* COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        assert(consensus.hashGenesisBlock ==
uint256S("0x000000000933ea01ad0ee984209779baaec3ced90fa3f408719526f8d77f4943"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
));

        vFixedSeeds.clear();
        vSeeds.clear();
        // nodes with support for servicebits filtering should be at the top
        vSeeds.emplace_back("testnet-seed.bitcoin.jonasschnelli.ch");
        vSeeds.emplace_back("seed.tbtc.petertodd.org");
        vSeeds.emplace_back("seed.testnet.bitcoin.sprovoost.nl");
        vSeeds.emplace_back("testnet-seed.bluematt.me"); // Just a static
list of stable node(s), only supports x9
```

```
        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,111);
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,196);
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,239);
        base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x35, 0x87, 0xCF};
        base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x35, 0x83, 0x94};

        bech32_hrp = "tb";

        vFixedSeeds = std::vector<SeedSpec6>(pnSeed6_test, pnSeed6_test +
ARRAYLEN(pnSeed6_test));

        fDefaultConsistencyChecks = false;
        fRequireStandard = false;
        fMineBlocksOnDemand = false;


        checkpointData = {
            {
                {546,
uint256S("000000002a936ca763904c3c35fce2f3556c559c0214345d31b1bcebf76acb70")}
,
            }
        };

        chainTxData = ChainTxData{
            // Data from rpc: getchaintxstats 4096
0000000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7cfc2b4c75
            /* nTime    */ 1531929919,
            /* nTxCount */ 19438708,
            /* dTxRate  */ 0.626
        };

        /* enable fallback fee on testnet */
        m_fallback_fee_enabled = true;
    }
};

/**
 * Regression test
 */
class CRegTestParams : public CChainParams {
public:
    explicit CRegTestParams(const ArgsManager& args) {
        strNetworkID = "regtest";
        consensus.nSubsidyHalvingInterval = 150;
        consensus.BIP16Exception = uint256();
        consensus.BIP34Height = 500; // BIP34 activated on regtest (Used in
functional tests)
        consensus.BIP34Hash = uint256();
        consensus.BIP65Height = 1351; // BIP65 activated on regtest (Used in
functional tests)
        consensus.BIP66Height = 1251; // BIP66 activated on regtest (Used in
functional tests)
        consensus.powLimit =
uint256S("7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 14 * 24 * 60 * 60; // two weeks
        consensus.nPowTargetSpacing = 10 * 60;
        consensus.fPowAllowMinDifficultyBlocks = true;
        consensus.fPowNoRetargeting = true;
        consensus.nRuleChangeActivationThreshold = 108; // 75% for testchains
```

```
        consensus.nMinerConfirmationWindow = 144; // Faster than normal for
regtest (144 instead of 2016)
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
0;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
Consensus::BIP9Deployment::ALWAYS_ACTIVE;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;

        // The best chain should have at least this much work.
        consensus.nMinimumChainWork = uint256S("0x00");

        // By default assume that the signatures in ancestors of this block
are valid.
        consensus.defaultAssumeValid = uint256S("0x00");

        pchMessageStart[0] = 0xfa;
        pchMessageStart[1] = 0xbf;
        pchMessageStart[2] = 0xb5;
        pchMessageStart[3] = 0xda;
        nDefaultPort = 18444;
        nPruneAfterHeight = 1000;
        m_assumed_blockchain_size = 0;
        m_assumed_chain_state_size = 0;

        UpdateVersionBitsParametersFromArgs(args);

        genesis = CreateGenesisBlock(1296688602, 2, 0x207fffff, 1, 50 *
COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        assert(consensus.hashGenesisBlock ==
uint256S("0x0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
));

        vFixedSeeds.clear(); //!< Regtest mode doesn't have any fixed seeds.
        vSeeds.clear();      //!< Regtest mode doesn't have any DNS seeds.

        fDefaultConsistencyChecks = true;
        fRequireStandard = false;
        fMineBlocksOnDemand = true;

        checkpointData = {
            {
                {0,
uint256S("0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206")}
,
            }
        };

        chainTxData = ChainTxData{
```

```
            0,
            0,
            0
        };

        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,111);
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,196);
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,239);
        base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x35, 0x87, 0xCF};
        base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x35, 0x83, 0x94};

        bech32_hrp = "bcrt";

        /* enable fallback fee on regtest */
        m_fallback_fee_enabled = true;
    }

    /**
     * Allows modifying the Version Bits regtest parameters.
     */
    void UpdateVersionBitsParameters(Consensus::DeploymentPos d, int64_t
nStartTime, int64_t nTimeout)
    {
        consensus.vDeployments[d].nStartTime = nStartTime;
        consensus.vDeployments[d].nTimeout = nTimeout;
    }
    void UpdateVersionBitsParametersFromArgs(const ArgsManager& args);
};

void CRegTestParams::UpdateVersionBitsParametersFromArgs(const ArgsManager&
args)
{
    if (!args.IsArgSet("-vbparams")) return;

    for (const std::string& strDeployment : args.GetArgs("-vbparams")) {
        std::vector<std::string> vDeploymentParams;
        boost::split(vDeploymentParams, strDeployment,
boost::is_any_of(":"));
        if (vDeploymentParams.size() != 3) {
            throw std::runtime_error("Version bits parameters malformed,
expecting deployment:start:end");
        }
        int64_t nStartTime, nTimeout;
        if (!ParseInt64(vDeploymentParams[1], &nStartTime)) {
            throw std::runtime_error(strprintf("Invalid nStartTime (%s)",
vDeploymentParams[1]));
        }
        if (!ParseInt64(vDeploymentParams[2], &nTimeout)) {
            throw std::runtime_error(strprintf("Invalid nTimeout (%s)",
vDeploymentParams[2]));
        }
        bool found = false;
        for (int j=0; j < (int)Consensus::MAX_VERSION_BITS_DEPLOYMENTS; ++j)
{
            if (vDeploymentParams[0] == VersionBitsDeploymentInfo[j].name) {
                UpdateVersionBitsParameters(Consensus::DeploymentPos(j),
nStartTime, nTimeout);
                found = true;
                LogPrintf("Setting version bits activation parameters for %s
to start=%ld, timeout=%ld\n", vDeploymentParams[0], nStartTime, nTimeout);
                break;
```

```
            }
        }
        if (!found) {
            throw std::runtime_error(strprintf("Invalid deployment (%s)",
vDeploymentParams[0]));
        }
    }
}

static std::unique_ptr<const CChainParams> globalChainParams;

const CChainParams &Params() {
    assert(globalChainParams);
    return *globalChainParams;
}

std::unique_ptr<const CChainParams> CreateChainParams(const std::string&
chain)
{
    if (chain == CBaseChainParams::MAIN)
        return std::unique_ptr<CChainParams>(new CMainParams());
    else if (chain == CBaseChainParams::TESTNET)
        return std::unique_ptr<CChainParams>(new CTestNetParams());
    else if (chain == CBaseChainParams::REGTEST)
        return std::unique_ptr<CChainParams>(new CRegTestParams(gArgs));
    throw std::runtime_error(strprintf("%s: Unknown chain %s.", __func__,
chain));
}

void SelectParams(const std::string& network)
{
    SelectBaseParams(network);
    globalChainParams = CreateChainParams(network);
}
```

Prilog P.4.3. Datoteka `src/chainparams.cpp` za rudarenje FERITcoin-a

```
// Copyright (c) 2010 Satoshi Nakamoto
// Copyright (c) 2009-2017 The Feritcoin Core developers
// Distributed under the MIT software license, see the accompanying
// file COPYING or http://www.opensource.org/licenses/mit-license.php.

#include <chainparams.h>
#include <consensus/merkle.h>

#include <tinyformat.h>
#include <util.h>
#include <utilstrencodings.h>

#include <assert.h>

#include <chainparamsseeds.h>

#include <arith_uint256.h>

static CBlock CreateGenesisBlock(const char* pszTimestamp, const CScript&
genesisOutputScript, uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t
nVersion, const CAmount& genesisReward)
{
    CMutableTransaction txNew;
    txNew.nVersion = 1;
    txNew.vin.resize(1);
    txNew.vout.resize(1);
    txNew.vin[0].scriptSig = CScript() << 486604799 << CScriptNum(4) <<
std::vector<unsigned char>((const unsigned char*)pszTimestamp, (const
unsigned char*)pszTimestamp + strlen(pszTimestamp));
    txNew.vout[0].nValue = genesisReward;
    txNew.vout[0].scriptPubKey = genesisOutputScript;

    CBlock genesis;
    genesis.nTime    = nTime;
    genesis.nBits    = nBits;
    genesis.nNonce   = nNonce;
    genesis.nVersion = nVersion;
    genesis.vtx.push_back(MakeTransactionRef(std::move(txNew)));
    genesis.hashPrevBlock.SetNull();
    genesis.hashMerkleRoot = BlockMerkleRoot(genesis);
    return genesis;
}

/**
```

```
 * Build the genesis block. Note that the output of its generation
 * transaction cannot be spent since it did not originally exist in the
 * database.
 *
 * CBlock(hash=000000000019d6, ver=1, hashPrevBlock=00000000000000,
hashMerkleRoot=4a5e1e, nTime=1231006505, nBits=1d00ffff, nNonce=2083236893,
vtx=1)
 *   CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
 *     CTxIn(COutPoint(000000, -1), coinbase
04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6
f72206f6e206272696e6b206f66207365636f6e642061696c6f757420666f722062616e6b73
)
 *     CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
 *   vMerkleTree: 4a5e1e
 */
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_t
nBits, int32_t nVersion, const CAmount& genesisReward)
{
    const char* pszTimestamp = "Vecernji list 22/June/2023 Tvrtka OceanGate
objavila: Vjerujemo da su svi putnici poginuli";
    const CScript genesisOutputScript = CScript() <<
ParseHex("04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb64
9f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f") <<
OP_CHECKSIG;
    return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime,
nNonce, nBits, nVersion, genesisReward);
}


void CChainParams::UpdateVersionBitsParameters(Consensus::DeploymentPos d,
int64_t nStartTime, int64_t nTimeout)
{
    consensus.vDeployments[d].nStartTime = nStartTime;
    consensus.vDeployments[d].nTimeout = nTimeout;
}


/**
 * Main network
 */
/**
 * What makes a good checkpoint block?
 * + Is surrounded by blocks with reasonable timestamps
 *   (no blocks before with a timestamp after, none after with
 *    timestamp before)
 * + Contains no strange transactions
 */


class CMainParams : public CChainParams {
public:
    CMainParams() {
        strNetworkID = "main";
```

```
        consensus.nSubsidyHalvingInterval = 210000;
        consensus.BIP16Height = 0;
        consensus.BIP34Height = 0;
        consensus.BIP34Hash =
uint256S("0x000000000000024b89b42a942fe0d9fea3bb44ab7bd1b19115dd6a759c0808b8"
);
        consensus.BIP65Height = 0;
        consensus.BIP66Height = 0;
        consensus.powLimit =
uint256S("0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 3.5 * 24 * 60 * 60; // 3.5 dana
        consensus.nPowTargetSpacing = 2.5 * 60;// 2.5min
        consensus.fPowAllowMinDifficultyBlocks = false;
        consensus.fPowNoRetargeting = false;
        consensus.nRuleChangeActivationThreshold = 1916; // 95% of 2016
        consensus.nMinerConfirmationWindow = 2016; // nPowTargetTimespan /
nPowTargetSpacing
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
1199145601; // January 1, 2008
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
1230767999; // December 31, 2008


        // Deployment of BIP68, BIP112, and BIP113.
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime =
1462060800; // May 1st, 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
1493596800; // May 1st, 2017


        // Deployment of SegWit (BIP141, BIP143, and BIP147)
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
1479168000; // November 15th, 2016.
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
1510704000; // November 15th, 2017.


        // The best chain should have at least this much work.
        consensus.nMinimumChainWork = uint256S("0x00");


        // By default assume that the signatures in ancestors of this block
are valid.
        consensus.defaultAssumeValid =
uint256S("0x00000000000000000005214481d2d96f898e3d5416e43359c145944a909d242e0"
); //506067


        /**
         * The message start string is designed to be unlikely to occur in
normal data.
         * The characters are rarely used upper ASCII, not valid as UTF-8,
and produce
```

```
         * a large 32-bit integer with any alignment.
         */
        pchMessageStart[0] = 0x32;
        pchMessageStart[1] = 0xd0;
        pchMessageStart[2] = 0x22;
        pchMessageStart[3] = 0xf8;
        nDefaultPort = 9191;
        nPruneAfterHeight = 100000;


        genesis = CreateGenesisBlock(1687613705, 127241, 0x1e00ffff, 1, 50 *
COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        /*
        for(; UintToArith256(genesis.GetHash()) >
UintToArith256(consensus.powLimit); genesis.nNonce++){
            if (genesis.nNonce % 1000 == 0){
                printf("Nonce: %d\n", genesis.nNonce);
                printf("Hash: %s\n", genesis.GetHash().ToString().c_str());
            }
        }
        printf("Genesis block: %s\n Merkle root: %s\n Nonce: %d\n",
genesis.GetHash().ToString().c_str(),

genesis.hashMerkleRoot.ToString().c_str(),

genesis.nNonce);
        */
        assert(consensus.hashGenesisBlock ==
uint256S("0x000054cef8151e510afe2f8c7d5b57a6565767d6e8d1c1cb8da6c988a9474e14"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0xf369bd39929aed314d5635ce6a65419457e59534a74ee3f5e8fb28505d455a38"
));


        // Note that of those which support the service bits prefix, most
only support a subset of
        // possible options.
        // This is fine at runtime as we'll fall back to using them as a
oneshot if they dont support the
        // service bits we want, but we should get them updated to support
all service bits wanted by any
        // release ASAP to avoid it where possible.


        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,95);
//95 stand for f (as feritcoin) according to
https://en.bitcoin.it/wiki/List_of_address_prefixes
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,122);
//as r
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,33);
//3 as E
        base58Prefixes[EXT_PUBLIC_KEY] = {0x27, 0x88, 0xB2, 0x1E};
        base58Prefixes[EXT_SECRET_KEY] = {0x27, 0x88, 0xAD, 0xE4};
```

```
        bech32_hrp = "bc";

        vFixedSeeds = std::vector<SeedSpec6>(pnSeed6_main, pnSeed6_main +
ARRAYLEN(pnSeed6_main));

        fDefaultConsistencyChecks = false;
        fRequireStandard = true;
        fMineBlocksOnDemand = false;

        checkpointData = {
            {
            }
        };

        chainTxData = ChainTxData{
            // Data as of block
000000000000000002d6cca6761c99b3c2e936f9a0e304b7c7651a993f461de (height
506081).
            1516903077, // * UNIX timestamp of last known number of
transactions
            295363220,  // * total number of transactions between genesis and
that timestamp
                        //   (the tx=... number in the SetBestChain debug.log
lines)
            3.5         // * estimated number of transactions per second
after that timestamp
        };
    }
};

/**
 * Testnet (v3)
 */
class CTestNetParams : public CChainParams {
public:
    CTestNetParams() {
        strNetworkID = "test";
        consensus.nSubsidyHalvingInterval = 210000;
        consensus.BIP16Height = 514; //
00000000040b4e986385315e14bee30ad876d8b47f748025b26683116d21aa65
        consensus.BIP34Height = 21111;
        consensus.BIP34Hash =
uint256S("0x0000000023b3a96d3484e5abb3755c413e7d41500f8e2a5c3f0dd01299cd8ef8"
);
        consensus.BIP65Height = 581885; //
00000000007f6655f22f98e72ed80d8b06dc761d5da09df0fa1dc4be4f861eb6
        consensus.BIP66Height = 330776; //
000000002104c8c45e99a8853285a3b592602a3ccde2b832481da85e9e4ba182
```

```
        consensus.powLimit =
uint256S("0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 14 * 24 * 60 * 60;
        consensus.nPowTargetSpacing = 10 * 60;
        consensus.fPowAllowMinDifficultyBlocks = true;
        consensus.fPowNoRetargeting = false;
        consensus.nRuleChangeActivationThreshold = 1512; // 75% for
testchains
        consensus.nMinerConfirmationWindow = 2016; // nPowTargetTimespan /
nPowTargetSpacing
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
1199145601; // January 1, 2008
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
1230767999; // December 31, 2008

        // Deployment of BIP68, BIP112, and BIP113.
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime =
1456790400; // March 1st, 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
1493596800; // May 1st, 2017

        // Deployment of SegWit (BIP141, BIP143, and BIP147)
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
1462060800; // May 1st 2016
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
1493596800; // May 1st 2017

        // The best chain should have at least this much work.
        consensus.nMinimumChainWork = uint256S("0x00");

        // By default assume that the signatures in ancestors of this block
are valid.
        consensus.defaultAssumeValid =
uint256S("0x0000000002e9e7b00e1f6dc5123a04aad68dd0f0968d8c7aa45f6640795c37b1"
); //1135275

        pchMessageStart[0] = 0x0b;
        pchMessageStart[1] = 0x11;
        pchMessageStart[2] = 0x09;
        pchMessageStart[3] = 0x07;
        nDefaultPort = 19191;
        nPruneAfterHeight = 1000;

        genesis = CreateGenesisBlock(1687613705, 2832037265, 0x1d00ffff, 1,
50 * COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
```

```
        assert(consensus.hashGenesisBlock ==
uint256S("0x994cb867d95f6c53fc42498ab8dc519bb662b510de1954923eb429b1de6954cb"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0xf369bd39929aed314d5635ce6a65419457e59534a74ee3f5e8fb28505d455a38"
));

        vFixedSeeds.clear();
        vSeeds.clear();
        // nodes with support for servicebits filtering should be at the top
        vSeeds.emplace_back("testnet-seed.feritcoin.jonasschnelli.ch");
        vSeeds.emplace_back("seed.tbtc.petertodd.org");
        vSeeds.emplace_back("seed.testnet.feritcoin.sprovoost.nl");
        vSeeds.emplace_back("testnet-seed.bluematt.me"); // Just a static
list of stable node(s), only supports x9

        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,111);
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,196);
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,239);
        base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x35, 0x87, 0xCF};
        base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x35, 0x83, 0x94};

        bech32_hrp = "tb";

        vFixedSeeds = std::vector<SeedSpec6>(pnSeed6_test, pnSeed6_test +
ARRAYLEN(pnSeed6_test));

        fDefaultConsistencyChecks = false;
        fRequireStandard = false;
        fMineBlocksOnDemand = false;


        checkpointData = {
            {
                {546,
uint256S("000000002a936ca763904c3c35fce2f3556c559c0214345d31b1bcebf76acb70")}
,
            }
        };

        chainTxData = ChainTxData{
            // Data as of block
000000000000033cfa3c975eb83ecf2bb4aaedf68e6d279f6ed2b427c64caff9 (height
1260526)
            1516903490,
            17082348,
            0.09
        };
```

```
    }
};

/**
 * Regression test
 */
class CRegTestParams : public CChainParams {
public:
    CRegTestParams() {
        strNetworkID = "regtest";
        consensus.nSubsidyHalvingInterval = 150;
        consensus.BIP16Height = 0; // always enforce P2SH BIP16 on regtest
        consensus.BIP34Height = 100000000; // BIP34 has not activated on
regtest (far in the future so block v1 are not rejected in tests)
        consensus.BIP34Hash = uint256();
        consensus.BIP65Height = 1351; // BIP65 activated on regtest (Used in
rpc activation tests)
        consensus.BIP66Height = 1251; // BIP66 activated on regtest (Used in
rpc activation tests)
        consensus.powLimit =
uint256S("00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
        consensus.nPowTargetTimespan = 14 * 24 * 60 * 60; // two weeks
        consensus.nPowTargetSpacing = 10 * 60;
        consensus.fPowAllowMinDifficultyBlocks = true;
        consensus.fPowNoRetargeting = true;
        consensus.nRuleChangeActivationThreshold = 108; // 75% for testchains
        consensus.nMinerConfirmationWindow = 144; // Faster than normal for
regtest (144 instead of 2016)
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].bit = 28;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nStartTime =
0;
        consensus.vDeployments[Consensus::DEPLOYMENT_TESTDUMMY].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].bit = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nStartTime = 0;
        consensus.vDeployments[Consensus::DEPLOYMENT_CSV].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].bit = 1;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nStartTime =
Consensus::BIP9Deployment::ALWAYS_ACTIVE;
        consensus.vDeployments[Consensus::DEPLOYMENT_SEGWIT].nTimeout =
Consensus::BIP9Deployment::NO_TIMEOUT;

        // The best chain should have at least this much work.
        consensus.nMinimumChainWork = uint256S("0x00");

        // By default assume that the signatures in ancestors of this block
are valid.
        consensus.defaultAssumeValid = uint256S("0x00");
```

```
        pchMessageStart[0] = 0xfa;
        pchMessageStart[1] = 0xbf;
        pchMessageStart[2] = 0xb5;
        pchMessageStart[3] = 0xda;
        nDefaultPort = 18444;
        nPruneAfterHeight = 1000;

        genesis = CreateGenesisBlock(1296688602, 2, 0x207fffff, 1, 50 *
COIN);
        consensus.hashGenesisBlock = genesis.GetHash();
        assert(consensus.hashGenesisBlock ==
uint256S("0x0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206"
));
        assert(genesis.hashMerkleRoot ==
uint256S("0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
));

        vFixedSeeds.clear(); //!< Regtest mode doesn't have any fixed seeds.
        vSeeds.clear();      //!< Regtest mode doesn't have any DNS seeds.

        fDefaultConsistencyChecks = true;
        fRequireStandard = false;
        fMineBlocksOnDemand = true;

        checkpointData = {
            {
                {0,
uint256S("0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206")}
,
            }
        };

        chainTxData = ChainTxData{
            0,
            0,
            0
        };

        base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,111);
        base58Prefixes[SCRIPT_ADDRESS] = std::vector<unsigned char>(1,196);
        base58Prefixes[SECRET_KEY] =     std::vector<unsigned char>(1,239);
        base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x35, 0x87, 0xCF};
        base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x35, 0x83, 0x94};

        bech32_hrp = "bcrt";
    }
```

```
};

static std::unique_ptr<CChainParams> globalChainParams;

const CChainParams &Params() {
    assert(globalChainParams);
    return *globalChainParams;
}

std::unique_ptr<CChainParams> CreateChainParams(const std::string& chain)
{
    if (chain == CBaseChainParams::MAIN)
        return std::unique_ptr<CChainParams>(new CMainParams());
    else if (chain == CBaseChainParams::TESTNET)
        return std::unique_ptr<CChainParams>(new CTestNetParams());
    else if (chain == CBaseChainParams::REGTEST)
        return std::unique_ptr<CChainParams>(new CRegTestParams());
    throw std::runtime_error(strprintf("%s: Unknown chain %s.", __func__,
chain));
}

void SelectParams(const std::string& network)
{
    SelectBaseParams(network);
    globalChainParams = CreateChainParams(network);
}

void UpdateVersionBitsParameters(Consensus::DeploymentPos d, int64_t
nStartTime, int64_t nTimeout)
{
    globalChainParams->UpdateVersionBitsParameters(d, nStartTime, nTimeout);
}
```

Prilog P.4.4. Datoteka `mining.sh`

```
echo "Sarting mining..."
start=$(date +%s)
curr_block_start=$(date +%s)
for i in {1..50000}
do
  MINE=$(src/bitcoin-cli generate 1)
  len=`expr length "$MINE"`
  if [[ len -gt 5 ]]
  then
    curr_block_end=$(date +%s)
    seconds=$((curr_block_end-curr_block_start))
    curr_block_start=$(date +%s)
    echo "$MINE"
    echo "Block time: $seconds sec"
  fi
done
end=$(date +%s)
seconds=$((end-start))
echo "Mining time: $seconds sec"
```